SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij automobilskog računarstva i komunikacija

DETEKCIJA VOZILA U OKRUŽENJU AUTONOMNOG VOZILA U SVRHU UPOZORENJA NA POTENCIJALNU KOLIZIJU

Diplomski rad

Mario Gluhaković

Osijek, 2019.

Sadržaj

1. UVOD

Povijest razvoja automobila počinje od potrebe za prijevoznim sredstvom koje će se samo pokretati, bez uporabe ljudske ili životinjske snage. Razvoj takvog prijevoznog sredstva omogućio je izum parnog stroja, pa je 1769. godine u Francuskoj N. J. Cugnot konstruirao prvi automobil na parni pogon. Međutim, ovakvi automobili nisu bili efikasni, jer su se kretali brzinom od 5 km/h i morali su se zaustavljati svakih 10 do 15 minuta kako bi postigli potreban tlak pare za daljnje pokretanje. Godine 1867. N. A. Otto je konstruirao četverotaktni plinski motor koji je kao gorivo koristio prirodni plin ili plin dobiven iz nafte ili ugljena. Razvojem motora s unutarnjim izgaranjem ubrzao se razvoj automobila, a prve automobile na takav pogon konstruirali su K. Benz i G. Daimler. Daljnjim razvojem dolazi do poboljšanja ovakvih pogona promjenom načina paljenja smjese goriva i zraka, pa je Maybach uveo paljenje pomoću užarene cijevi, dok je Bosch uveo napredniji sustav za paljenje s pomoću električne svjećice. S vremenom su motori u vozilima postali snažniji, lakši, ekonomičniji, a samim automobilima je povećana brzina. Kao izvor pogona teretnih vozila ustalio se Dieselov motor, dok se za pogon automobila koristio *Ottov* (benzinski) motor. U današnje vrijeme povećanje sigurnosti u automobilima, te smanjenje potrošnje goriva i poboljšanje njihovih ekoloških značaja glavni su zahtjevi koji su stavljeni pred automobilsku industriju. Zbog poboljšanja ekološke efikasnosti vozila dolazi do razvoja i proizvodnje hibridnih i električnih vozila koja smanjuju zagađenje ispušnim plinovima.

U zadnjem desetljeću dolazi do snažnog razvoja autonomnih vozila. Autonomna vozila su vozila koja su sposobna detektirati objekte u svojoj okolinu i kretati se bez ili s jako malo ljudskog utjecaja. Autonomna vozila kombiniraju podatke s različitih senzora kako bi mogli percipirati svoju okolinu. Neki od senzora koji se koriste su: kamera, radar, lidar, ultrazvučni senzori i *GPS* (engl. *Global Positioning System*). Napredni sustavi upravljanja koriste informacije sa senzora da bi odredili putanje kretanja i uočili moguće prepreke.

Napredni sustavi za pomoć vozaču (engl. *Advanced Driver-Assistance Systems - ADAS*) su elektronički sustavi koji pomažu vozaču tijekom vožnje. Namjena ovakvih sustava je pomoć vozaču pri vožnji, povećanje sigurnosti putnika i općenito sigurnosti na cesti. Većina prometnih nesreća događa se zbog ljudske pogreške, te je glavna zadaća automatiziranih sustava koje *ADAS* pruža vozilu smanjenje broja smrtnih slučajeva na cestama, tako da se vjerojatnost ljudske pogreške što više smanji. Neka od uobičajenih *ADAS* sustava su: prilagodljivi tempomat,sustav za izbjegavanje sudara, sustav upozorenja o napuštanju trake, automatsko centriranje trake, *ABS*, *ESP* i dr.

SAE International je definirala 6 razina autonomije za vozila. Nulta razina (engl. *No Driving Automation*) je razina kod koje se automobil u potpunosti ručno kontrolira. Prva razina (engl. *Driver Assistance*) je najniža razina autonomije. Vozila na ovoj razini imaju jednostavne sustave za pomoć vozaču, kao što su kontrola ubrzanja ili upravljanja. Druga razina (engl. *Partial Driving Automation*) podrazumijeva *ADAS* sustave za pomoć vozaču. Vozilo može samostalno upravljati, ubrzavati i usporavati. Međutim, tijekom čitave vožnje vozač se mora nalaziti u vozilu, te može preuzeti kontrolu nad vozilom u svakom trenutku. Razina tri (engl. *Conditional Driving Automation*) donosi mogućnost detekcije objekata u okolini vozila. U ovom slučaju vozila mogu donositi odluke samostalno, kao što je obilazak sporog vozila, međutim ovi sustavi još uvijek zahtijevaju stalni nadzor vozača. Na razini četiri (engl. *High Driving Automation*) vozilo može voziti u autonomnom načinu, ali samo unutar ograničenog područja, te vozač uvijek ima mogućnost ručnog upravljanja. Najveća razina autonomije je peta razina (engl. *Full Driving Automation*). Vozila na petoj razini ne zahtijevaju ljudsku prisutnost, te nemaju upravljače ili papučice za kontroliranje kočenja i ubrzanja.

U ovom radu razvijen je i opisan algoritam za prepoznavanje vozila u okolini autonomnog vozila i procjenu udaljenosti koji je implementiran u *ROSu* (engl. *Robot Operating System*) [1]. U radu je korišten *Yolo v2* (engl. *You only look once*) [2] algoritam koji je zasnovan na konvolucijskim neuronskim mrežama (engl. *Convolutional neural network – CNN*), pomoću kojih algoritam prepoznaje vozila u video sekvenci s kamere. Za implementaciju Yolo v2 algoritma u *ROS* koristi se *Darknet ROS* čvor. *Yolo v2* algoritam parametre detektiranih vozila prosljeđuje *ROS* čvoru koji vrši određivanje udaljenosti detektiranih vozila. Algoritam određuje udaljenost detektiranih vozila da bih se točno znala pozicija okolnih vozila u odnosu na autonomno vozilo, te da bi se tako predvidjele moguće kolizije s autonomnim vozilom. Rezultati algoritma su detekcija vrste vozila i udaljenost u odnosu na autonomno vozilo.

U drugom poglavlju opisana su postojeća rješenja koja se bave detekcijom vozila u okolini autonomnog vozila i određivanje pozicija i trajektorija kretanja detektiranih vozila. Treće poglavlje opisuje rješenje za detekciju i određivanje udaljenosti vozila u okolini autonomnog vozila razvijenog u sklopu ovog diplomskog rada. Četvrto poglavlje prikazuje evaluaciju razvijenog rješenja. U petom poglavlju su predstavljeni zaključci rada.

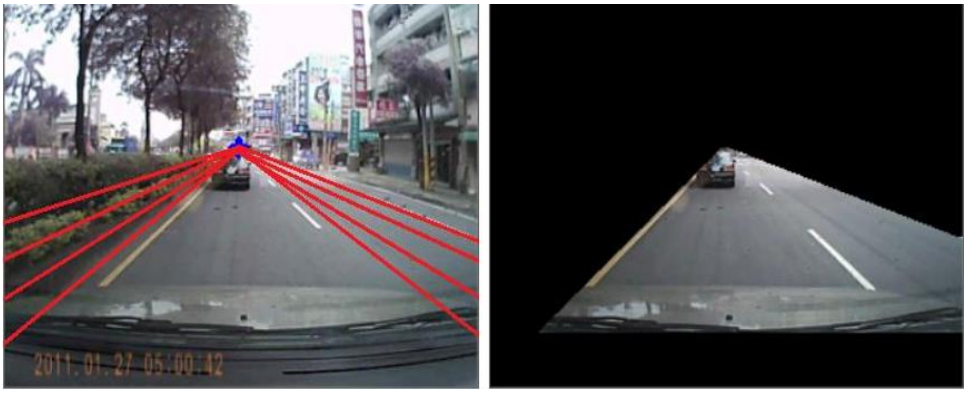
2. PREGLED POSTOJEĆIH RJEŠENJA ZA DETEKCIJU VOZILA I PROCJENU UDALJENOSTI U SVRHU OTKRIVANJA POTENCIJALNE KOLIZIJE

Postoje mnoga rješenja za detekciju vozila i procjenu udaljenosti pomoću različitih senzora na autonomnom vozilu. U ovom poglavlju je dan pregled postojećih rješenja i njihovih nedostataka koji mogu utjecati na točnost detekcije i procjenu udaljenosti detektiranog vozila, u odnosu na autonomno vozilo.

U radu [3], pomoću kamere se vrši detekcija i procjena udaljenosti od vozila u okolini, a algoritam se sastoji iz dva glavna koraka. Prvi korak je detekcija i praćenje vozila, a drugi korak je određivanje udaljenosti za svako vozilo. U prvom koraku ovoga algoritma se pomoću prilagodljivog klizećeg prozora biraju najbolji mogući kandidati prozori. Klizeći prozor u računalnom vidu je kvadratna površina koja prelazi (klizi) po slici. Razlika između klizećeg prozora i prilagodljivog klizećeg prozora je ta, da se veličina prilagodljivog klizećeg prozora tijekom iteracija mijenja, dok je kod običnog klizećeg prozora ona uvijek ista. Na svakom od tih prozora traže se značajke slične *Haarovim* značajkama [4], da bi se odredilo da li prozor sadrži objekt od interesa [5]. Neke od značajki koje koristi ovaj algoritam su sjena ispod prednjeg branika, zadnje gume na vozilu, vertikalni rubovi na krajevima vozila, horizontalni rubovi na sredini prednje i zadnje strane vozila i dr. Međutim, problem je što u nekim slučajevima ovaj algoritam daje pogrešne detekcije. Za poboljšanje točnosti prilikom detekcije predloženo je dodavanje više značajki koje detektiraju dodatne rubove na vozilu, koje bi smanjile broj pogrešnih detekcija. Za praćenje detektiranih vozila koristi se *Kalman* filtar [6]. S obzirom na to da za mjerenje udaljenosti visina vozila nije bitna, algoritam zanemaruje visinu vozila i pretpostavlja da je odnos visine i širine vozila uvijek konstantan. Za svaki detektirani i praćeni objekt vrši se određivanje udaljenosti, te su razvijena dva načina određivanja udaljenosti. Prvi način se zasniva na određivanju udaljenosti na osnovu širine detektiranog vozila. Da bi se udaljenost odredila na prvi način potrebno je poznavati stvarnu širinu vozila i specifikacije kamere. S obzirom na to da postoje razni tipovi vozila i njihova stvarna širina nije jednaka, dolazi do pogrešnog određivanja udaljenosti. U drugoj metodi, koristi se vertikalna pozicija detektiranog vozila na slici i pretpostavka da su udaljenost i vertikalna pozicija vozila u slici proporcionalne. Također, u ovoj metodi potrebno je pretpostaviti da je cesta ravna. Ako se zadovolje sve pretpostavke ova metoda se može koristiti, međutim ova metoda je osjetljiva i na jako male promjene boje u elementima slike. U konačnici, da bi se riješili nedostatci gore navedenih metoda za određivanje udaljenosti koristi se njihova kombinacija.

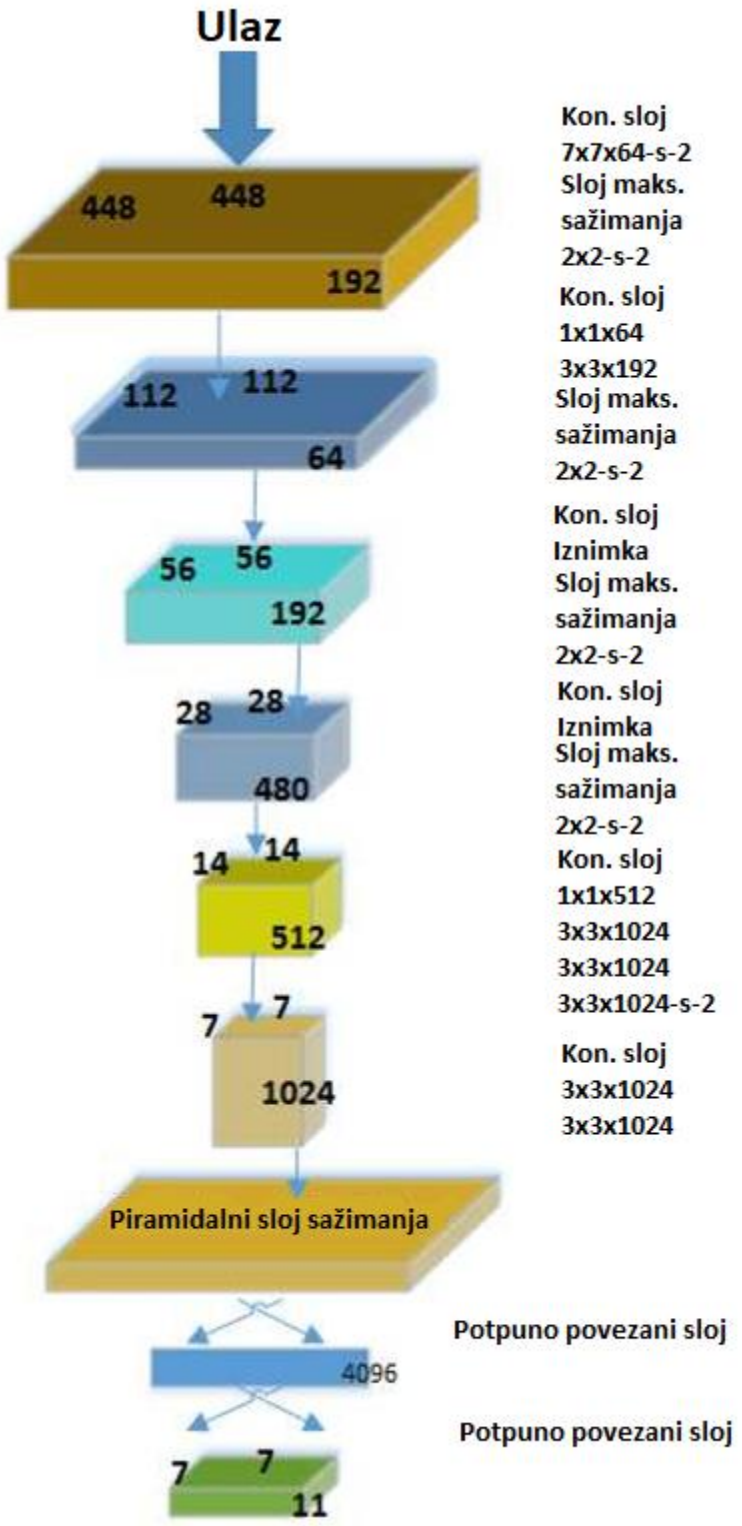
Slika 2.1. *Određivanje udaljenosti (a) na osnovu širine, (b) na osnovu pozicije* [3]

U radu [7] koristi se kamera za detekciju prednje strane vozila i procjenu udaljenosti do njega. Ovo rješenje je zasnovano na: detekciji rubova ceste, određivanju točke nestajanja (engl. *Vanishing point*) [8], segmentaciji ceste, detekciji vozila i izračunavanju udaljenosti do detektiranog vozila. Na početku, algoritam pomoću *Canny* detektora [9] rubova određuje sve rubove u slici. Smanjenje broja detektiranih linija se vrši pomoću *Houghove* transformacije [10], koja u svim detektiranim linijama pronalazi pravilne ravne linije i u konačnici se kao rezultat dobiva orijentacija ceste. S obzirom na to da je kamera postavljena na sredinu vjetrobranskog stakla, linije koje se nalaze na lijevoj strani slike s kamere orijentirane su prema desno, dok su linije koje se nalaz na desnoj strani orijentirane prema lijevo što se može vidjeti na slici 2.2. S obzirom na takvu orijentaciju linija, algoritam razdvaja sliku s kamere u lijevu i desnu polovinu sliku. Algoritam određuje kojoj polovini slike pripada koji element slike odnosno linija. Na ovaj način određuje se točka nestajanja, a to je zapravo točka u kojoj se presijeca većina linija iz obje polovine slike. Segmentacija ceste se obavlja tako što se detektirane linije povežu s točkom nestajanja. Pretpostavka je da će se vozila koja je potrebno detektirati ispred vozila na koji je postavljena kamera uvijek nalaziti u segmentiranom dijelu ceste. Vozilo unutar segmentiranog dijela ceste se određuje pomoću *Sobelovog* filtra [11] za detekciju rubova koji koristi 3x3 filtar. Na osnovu pozicije automobila i točke nestajanja određuje se udaljenost između detektiranog vozila i prednje strane vozila na koji je montirana kamera. Eksperimentalnim testiranjem došlo se do zaključka da je postotak točne detekcije vozila oko 78%.



Slika 2.2. *Segmentacija ceste i detekcija vozila* [7]

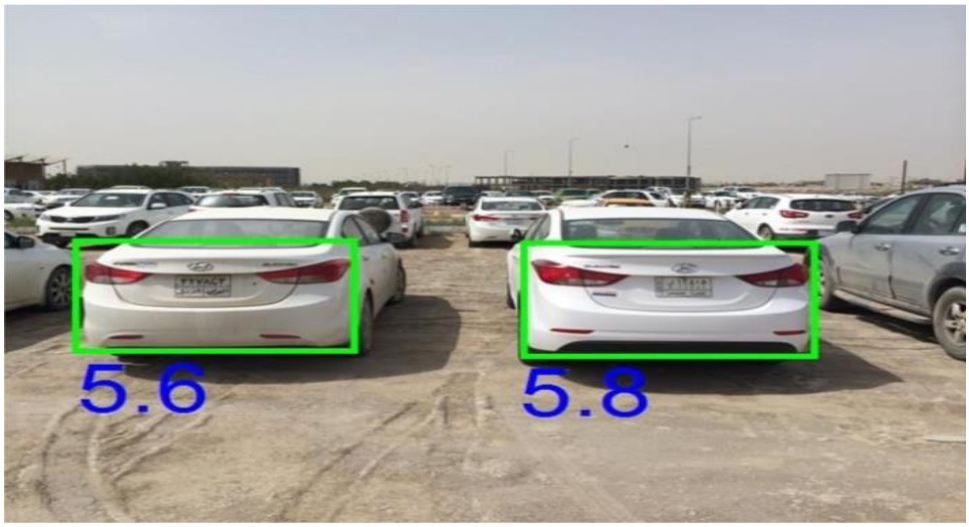
U radu [12], predložena je metoda detekcije i praćenja vozila u stvarnom vremenu korištenjem dubokih neuronskih mreža. U predloženoj metodi se ne određuju udaljenosti, ali se pokazuje da se pomoću dubokih neuronskih mreža može u stvarnom vremenu detektirati i pratiti vozila pomoću kamere. Ova metoda zasnovana je na dubokoj neuronskoj mreži koja se sastoji od: konvolucijskih slojeva, piramidalnog sloja sažimanja, slojeva iznimke, slojeva maksimalnog sažimanja i potpunog povezanih slojeva kao što je to prikazano na slici 2.3. Sloj iznimke je koristan u kontekstu lokalizacije i detekcije objekata [13], te se stoga u ovoj arhitekturi koriste četiri sloja iznimke. Prvi put ideja korištenja sloja iznimke je predložena u *GoogleNet* [14] mreži. Kao što se zna, potpuno povezani sloj zahtjeva fiksnu veličinu ulazne slike i stoga se koristi piramidalni sloj sažimanja koji može svaku ulaznu sliku da smanji na fiksnu veličinu. Na slici 2.3. može se vidjeti arhitektura duboke neuronske mreže koja se koristi u ovom radu. Prvi dio je 7x7 konvolucijski sloj s veličinom filtra 64. Nakon njega se nalazi sloj maksimalnog sažimanja koji smanjuje prostor značajki za sljedeći sloj. Sljedeći sloj je 1x1 konvolucijski sloj iza kojeg slijedi 3x3 konvolucijski sloj sa 192 filtera. U nastavku se nalaze slojevi iznimke i slojevi maksimalnog sažimanja koji rezultiraju izlazom veličine 14 x 14 x 512. Ostatak mreže čini 6 konvolucijskih slojeva i potpuno povezani sloj. Posljednji sloj mreže vrši procjenu da li se u slici nalazi vozilo i na kojim koordinatama se ono nalazi. Prethodno opisana neuronska mreža u svojim slojevima koristi ispravljenu linearnu aktivacijsku funkciju, dok u zadnjem sloju koristi linearnu aktivacijsku funkciju [15]. Korištena mreža je trenirana na *PASCAL VOC 2007* i *2012* bazi podataka. Nakon treniranja mreža je testirana, te je pokazano da mreža ima srednju preciznost od 80.5%.



Slika 2.3. *Arhitektura mreže* [12]

U radu [16] je predstavljen algoritam za detekciju vozila i određivanje udaljenosti na osnovu monokularne kamere. Ovaj algoritam zasnovan je na *Viola-Jones* algoritmu za detekciju objekata [17]. *Viola-Jones* algoritam je algoritam strojnog učenja za detekciju objekata pomoću Haarovih značajki. Predstavljeni algoritam koristi istrenirani *Viola-Jones*

algoritam koji se nalazi u *OpenCV* (engl. *Open source computer vision*) biblioteci [18]. Nakon što se vozilo detektira, određuje se udaljenost od kamere do detektiranog vozila uz pomoć parametara kamere, stvarne širine vozila i detektirane širine vozila u slici. Drugi problem, koji ovaj rad pokušava riješiti je zadržavanje vozila u traci i upozorenje na napuštanje vozne trake. U ovom dijelu algoritma vozna traka se određuje korištenjem *Houghove* transformacije i *Kalman* filtra. Nakon što se odrede linije koje omeđuju voznu traku, izračunava se sredina između njih te se na temelju izračunate sredine između traka određuje pravilna putanja vozila kako bi ono bilo uvijek na sredini vozne trake.



Slika 2.4. *Primjer rada algoritma* [16]

3. ALGORITAM ZA PROCJENU UDALJENOSTI VOZILA U OKOLINI AUTONOMNOG VOZILA

3.1. YOLO algoritam

Postoji nekoliko algoritama za detekciju objekata koji se mogu razdvojiti u dvije glavne grupe: algoritmi zasnovani na klasifikaciji i algoritmi zasnovani na regresiji. Algoritmi zasnovani na klasifikaciji rade u dva koraka. U prvom koraku se iz slike odabiru interesna područja. U drugom koraku se ta područja korištenjem konvolucijskih neuronskih mreža klasificiraju. Ovakva rješenja mogu biti jako spora jer se predikcija mora obavljati za sva odabrana područja. Jedan od poznatih primjera ovakvog algoritma je *RCNN* (engl. *Region-based convolutional neural network*) [19]. Algoritmi zasnovani na regresiji umjesto odabira interesnog područja rade predikciju klase i graničnog okvira za cijelu sliku u jednom prolasku kroz algoritam. Najpoznatiji algoritam ove vrste je *Yolo* algoritam, koji se uobičajeno koristi za detekciju objekata u stvarnom vremenu. *Yolo* algoritam je jedan od najučinkovitijih algoritama za detekciju objekata u stvarnom vremenu. *Yolo* algoritam se prvi put spominje 2015. godine u radu „You Only Look Once: Unified, Real-Time Object Detection“ [20]. Ovaj algoritam je postao popularan zbog svoje visoke preciznosti prilikom rada u stvarnom vremenu.

Yolo algoritam na ulazu prima sliku koju potom provlači kroz konvolucijsku neuronsku mrežu, koja na svom izlazu daje vektor graničnih okvira i klasu objekata koji se nalaze unutar graničnih okvira. Na početku se slika dijeli u mrežu ćelija veličine $S \times S$. U svakoj ćeliji se predviđa B graničnih okvira i C vjerojatnost pojavljivanja klase. Svaki granični okvir sastoji se od 5 komponenata: x, y, w, h i c. Koordinate x i y predstavljaju centar graničnog okvira koji je relativan u odnosu na centar ćelije u kojoj se objekt nalazi. Vrijednosti w i h su širina i visina graničnog okvira koje su relativne u odnosu na veličinu slike i njihova vrijednost je u rasponu od 0 do 1. Oznaka c je sigurnost predviđene klase unutar graničnog okvira. Ako se u graničnom okviru ne nalazi objekt vrijednost je 0, a ako se unutar graničnog okvir nalazi objekt vrijednost je jednaka *IoU* (engl. *Intersection over union*) [21]. Algoritam na osnovu vrijednosti c određuje da li unutar graničnog okvira postoji objekt. Svaka ćelija sadrži B predviđenih graničnih okvira. S obzirom na to da *Yolo v2* algoritam unutar jedne ćelije može da detektira do 5 različitih objekata za čitavu sliku imamo $S \times S \times B * 5$ predviđenih graničnih okvira.

IoU je jednostavan metrički način evaluacije algoritama koji kao izlaz daju predviđeni granični okvir. Da bi se koristio ovaj način evaluacije, potrebno je imati ručno označene

granične okvire temeljne istine (engl. *ground-truth bounding boxes*) na skupu za testiranje i predviđeni granični okvir koji dobijemo korištenjem našeg algoritma. Na slici 3.1. možemo vidjeti primjer na kojem je vidljiv okvir temeljne istine (zeleni okvir) i granični okvir koji je predvidio algoritam (crveni okvir), te je vidljivo da se ta dva okvira ne podudaraju u potpunosti.

Slika 3.1. *Prikaz okvira temeljne istine i predviđenog graničnog okvira [21]*

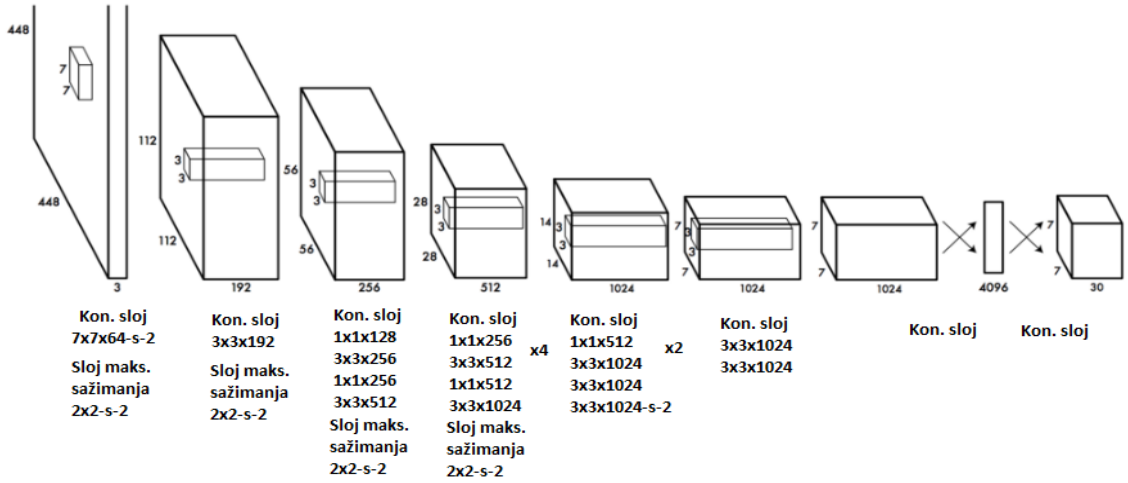
Da bismo znali koliko se ta dva područja podudaraju koristi se *IoU*. Za računanje *IoU* koristi se formula (3-1) [21]:

$$IoU = \frac{površina\ preklapanja\ detektiranog\ okvira\ i\ okvira\ temeljne\ istine}{površina\ unije\ detektiranog\ okvira\ i\ okvira\ temeljne\ istine} \tag{3-1}$$

Kao što se po formuli (3-1) može vidjeti, vrijednost *IoU* se dobije dijeljenjem vrijednosti površine preklapanja i površine unije okvira temeljne istine i predviđenog graničnog okvira.

Yolo konvolucijska neuronska mreža inspirirana je *GoogLeNet* neuronskom mrežom za klasifikaciju slika. *Yolo* mreža ima 24 konvolucijska sloja nakon kojih se nalaze 2 potpuno povezana sloja. Umjesto modula iznimke *Yolo* mreža koristi 1 x 1 slojeve smanjivanja nakon kojih slijede 3 x 3 konvolucijski slojevi. Na slici 3.2. se može vidjeti arhitektura *Yolo* konvolucijske neuronske mreže. Prva verzija *Yolo* algoritma je *Yolo v1* [23] algoritam, koji je imao nekoliko ograničenja. *Yolo v1* nije mogao pronaći male objekte ako se oni pojavljuju kao klaster, tj. niz malih objekata koji se preklapaju. Također, ovakva arhitektura je imala problem pri generalizaciji objekata ako je dimenzija slike različita od dimenzija slika koje su se koristile za treniranje. Treći i najveći problem je bio lokalizacija objekata u ulaznoj slici. S obzirom na sve prethodno navedene probleme, razvijena je nova verzija *Yolo* algoritma pod nazivom *YOLO9000* odnosno *Yolo v2* algoritam [23]. *Yolo v2* algoritam predstavljen je 2016.

godine. Da bi se riješili problemi u v1 verziji algoritma obavljene su određene promjene na *Yolo* algoritmu.



Slika 3.2. *Arhitektura Yolo konvolucijske neuronske mreže [22]*

U novu verziju je dodana normalizacija serije (engl. *Batch Normalization*), koja smanjuje pomicanje jedinične vrijednosti u skrivenom sloju i tako poboljšava stabilnost neuronske mreže. Ulazna veličina slike je povećana sa 224 x 224 na 448 x 448. Ovo povećanje ulazne veličine slike povećalo je srednju točnost za 4%. Jedna od prepoznatljivih izmjena koje je donio *Yolo v2* algoritam je uvođenje pristupa predviđanja graničnih okvira (engl. *Anchor Boxes*). Jedan od glavnih problema *Yolo v1* algoritma je bila detekcija malih objekata u slici, što je riješeno u *Yolo v2* algoritmu dijeljenjem ulazne slike u 13 x 13 mrežu ćelija. Ovaj način omogućuje *Yolo v2* algoritmu detekciju i lokalizaciju manjih objekata u slici. Problem detekcije objekata u slikama različitih rezolucija riješeno je treniranjem *Yolo v2* algoritma slikama različitih dimenzija između 320 x 320 i 608 x 608, što omogućuje mreži da nauči predviđati objekte u slikama različitih dimenzija sa što većom točnošću. *Yolo v2* koristi *Darknet 19* arhitekturu s 19 konvolucijskih slojeva, 5 slojeva maksimalnog sažimanja i *softmax* slojem za klasifikaciju objekata. *Darknet* [24] je *framework* za neuronske mreže napisan u *C* programskom jeziku [25] i *CUDA* [26]. *Darknet* je besplatni *framework* koji služi kao osnova za *Yolo* algoritam te se koristi kao platforma za treniranje i korištenje *Yolo* algoritma. Ovaj framework sadrži sve što je potrebno za korištenje *Yolo* algoritma.

Treća verzija *Yolo* algoritma donosi nekoliko promjena u odnosu na *Yolo v2* algoritam. *Yolo v3* [23] algoritam u zadnjem sloju koristi logički klasifikator za svaku klasu umjesto *softmax* sloja kao što je to bio slučaj kod *Yolo v2* algoritma. *Yolo v3* algoritam čini predviđanja na sličan način kao FPN (engl. *Feature Pyramid Networks*) [27], gdje kreiraju 3

predikcije za svaku lokaciju na ulaznoj slici. Na ovaj način dobiva se veća točnost na različitim skalama. *Yolo v3* algoritam koristi *Darknet-53* arhitekturu koja sadrži 53 konvolucijska sloja, te je mnogo dublja od *Darknet-19* arhitekture koju koristi *Yolo v2* algoritam. Ove promjene donose veću srednju točnost algoritmu, ali već zbog dubine mreže zahtjeva jače računalo za treniranje i pokretanje od *Yolo v2* algoritma. U ovom diplomskom radu, za detekciju objekata u slici korištena je *Yolo v2* verzija algoritma.

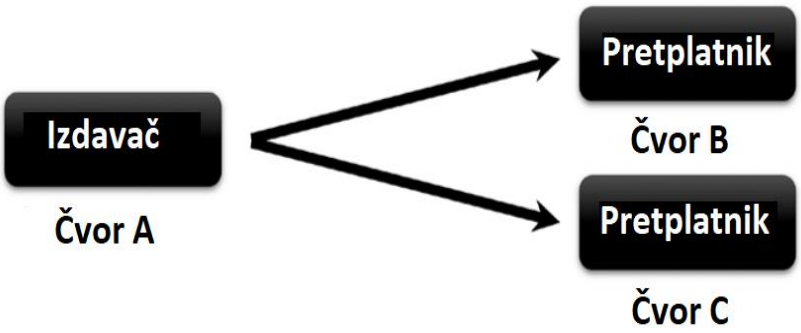
3.2. ROS

ROS je meta operacijski sustav za robote. On pruža usluge koje se očekuju od operacijskog sustava, nisku razinu upravljanja uređajima, implementaciju često korištenih funkcionalnosti, prosljeđivanje poruka između procesa i upravljanje paketima unutar sustava. On također pruža alate i biblioteke za pisanje, izgradnju i pokretanje koda na više različitih platformi. Meta operacijski sustav predstavlja sustav, koji obavlja procese kao što je planiranje, obavljanje i praćenje procesa i rukovanje iznimkama tako što inicijalizira virtualizacijski sloj između aplikacija i računalnih komponenti. *ROS* nije konvencionalni operacijski sustav kao što je *Windows, Linux* i slični, nego je on meta operacijski sustav koji se pokreće na nekom već postojećem operacijskom sustavu. Jedan od često korištenih primjera je instalacija *ROSa* na *Ubuntu* operacijskom sustavu. Tako se kreira simbioza *Ubuntu* operacijskog sustava i *ROSa* u kojoj *ROS* pruža potrebne funkcionalnosti za rad s robotskim aplikacijama, dok *Ubuntu* operacijski sustav pruža upravljanje procesima, datotečni sustav, grafičko korisničko sučelje i dr. Cilj *ROSa* je izgradnja razvojnog okruženja koji će pružiti razvoj robotske programske podrške na globalnoj razini. *ROS* za cilj ima maksimizirati ponovnu upotrebu napisanog koda. *ROS* pruža razne klijent biblioteke koje omogućavaju razvoj korištenjem raznih programskih jezika kao što su: *Python, C++, Lisp, JAVA i C#*.

S obzirom na to da je razvoj *ROSa* započeo 2007. godine, do danas je predstavljeno mnogo različitih verzija ovog operacijskog sustava. *ROS* verzije se osvježavaju dvaput godišnje. Podrška za svaku verziju *ROSa* je različita ali je uobičajeno da podrška traje dvije godine od prve objave verzije. U ovom radu korišten je *ROS* na *Ubuntu* operacijskom sustavu. Prilikom odabira verzije *ROSa* potrebno je paziti jer se za svaku distribuciju *Ubuntu* operacijskog sustava koristi druga verzija *ROSa*. U radu je korišten *Ubuntu 16.04 Xenial Xerus* operacijski sustav te je unutar njega implementirana *Kinetic Kame* distribucija ROSa.

ROS pruža standardne servise operacijskog sustava kao što je apstrakcija sklopovlja, programsku podršku za uređaje, implementaciju često korištenih značajki kao što su

prepoznavanje, mapiranje, planiranje putanja, prosljeđivanje poruka između procesa, upravljanje paketima, te biblioteke za razvoj. Da bi se mogao koristiti *ROS* potrebno je poznavati osnovne koncepte koji se koriste unutar njega. *Master* predstavlja glavni server za komunikaciju između čvorova. Komunikacija između čvorova bez *Mastera* nije moguća. Čvorovi (engl. *nodes*) predstavljaju jedinstvene procese koji se obavljaju. Svaki čvor ima ime, te više čvorova s različitim imenima može postojati unutar različitih imenskih prostora. Teme (engl. *topics*) se nazivaju sabirnice preko kojih čvorovi šalju i primaju poruke. Imena tema moraju biti jedinstvena unutar svog prostora imena. Za slanje poruke čvor mora objaviti temu, a za primanje poruke se mora pretplatiti na temu. Čvor također može oglašavati usluge (engl. *service*). Usluga predstavlja radnju koju čvor može preuzeti, a koja će imati jedan jedinstveni rezultat. Zbog toga se usluge često koriste za radnje koje imaju definirani početak i kraj. Čvorovi oglašavaju usluge i pozivaju usluge jedni od drugih. Čvorovi šalju i primaju podatke pomoću poruka (engl. *messages*). Na slici 3.3. prikazana je komunikacija tri čvora, od kojih čvor *A* oglašava uslugu, a čvorovi *B* i *C* se pretplaćuju na nju.



Slika 3.3. *Komunikacija tri čvora unutar ROSa [1]*

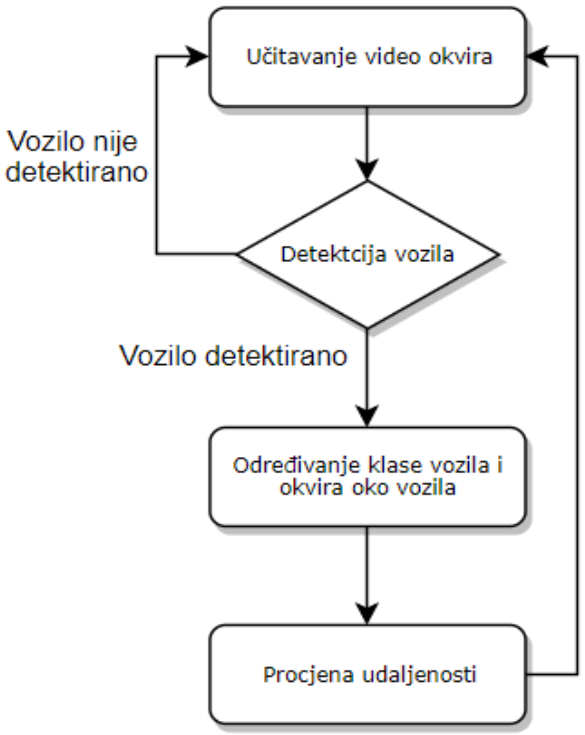
3.3. Opis rada vlastitog algoritma

Procjena pozicija (udaljenosti) drugih vozila u okolini autonomnog vozila, pomaže pri procjeni kretanja drugih vozila i procjeni hoće li doći do moguće kolizije između ta dva vozila, na osnovu njihove udaljenosti. Na temelju četiri kamere koje snimaju čitavo okruženje od 360 stupnjeva oko autonomnog vozila detektiraju se okolna vozila. Za detekciju vozila se koristi *Yolo v2* algoritam za detekciju objekata, koji je zasnovan na konvolucijskim neuronskim mrežama. *Yolo* algoritam je treniran na kreiranom skupu za treniranje koji je detaljnije opisan u poglavlju 3.3.1., te može detektirati 4 klase objekata: automobil, kombi, autobus i kamion. Algoritam koji je razvijen u sklopu ovog diplomskog rada je implementiran

unutar *ROSa*, unutar kojeg se vrši procjena udaljenosti detektiranog vozila. Za procjenu udaljenosti se koristi poseban *ROS* čvor koji prihvaća parametre proslijeđene od *Yolo* algoritma i pomoću njih vrši procjenu udaljenosti. Algoritam za procjenu udaljenosti podijeljen je u dva dijela:

1. Detekcija vozila u okolini pomoću *Yolo v2* algoritma
2. Procjena udaljenosti detektiranih vozila

Na slici 3.4. je prikazan dijagram algoritma za detekciju vozila u okolini autonomnog vozila i procjenu udaljenosti detektiranih vozila koji je kreiran u sklopu ovog diplomskog rada.



Slika 3.4. *Blok dijagram algoritma za procjenu udaljenosti detektiranih vozila u okolini*

3.3.1. Detekcija vozila u okolini autonomnog vozila

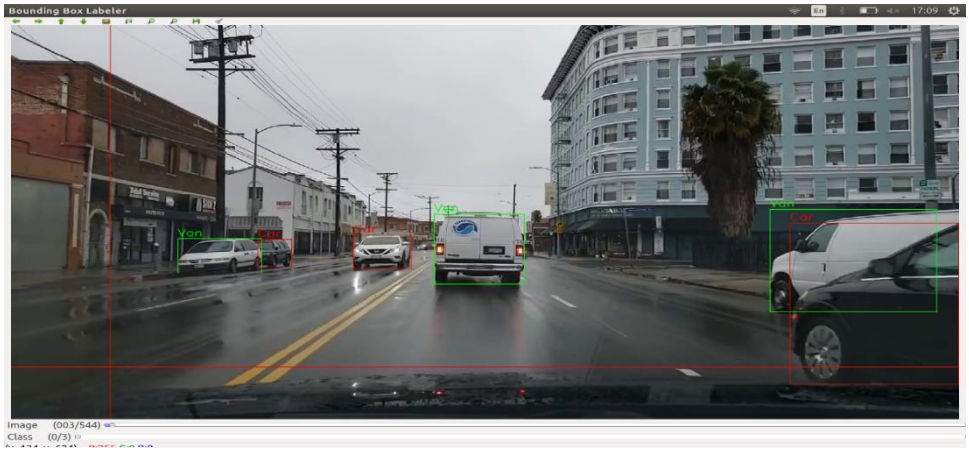
Kao što se moglo vidjeti u drugom poglavlju, najbitniji dio algoritma za procjenu udaljenosti je zapravo detekcija vozila u okolini autonomnog vozila. Da bi se što točnije procijenila udaljenost, potrebno je što preciznije detektirati vozila u slici koja dolazi s kamera. Detekcija vozila u slici koja dolazi s kamere se vrši pomoću *Yolo v2* algoritma. Da bi se *Yolo v2* algoritam mogao koristiti za detekciju željenih objekata, potrebno je obaviti određene promjene na mreži, kreirati skup za treniranje mreže i istrenirati mrežu na novom skupu. *Yolo v2* algoritam koji se koristi unutar ovog diplomskog rada, prenamijenjen je tako da detektira četiri klase objekata: automobil, kombi, autobus i kamion. Prvo je potrebno kreirati skup slika za treniranje mreže i označiti granične okvire na slikama za treniranje.

Slike za treniranje su prikupljene pomoću različitih izvora: *ImageNet* [28] baza slika, *PASCAL VOC* [29] baza slika, *KITTI* [30] baza slika i video okvira koji su nastali snimanjem prometa u gradu Osijeku. Promet u gradu Osijeku je sniman *GoPro Hero 5* kamerom. Nakon prikupljanja slika, na svakoj slici su označeni granični okvir objekata koje je potrebno detektirati. Označena baza slika za treniranje se nalazi u prilogu P3.1. na priloženom DVDu. U tablici 3.1. se nalazi prikaz broja objekata unutar baze podataka za treniranje mreže.

Tablica 3.1. *Broj objekata po klasama za treniranje Yolo v2 mreže*

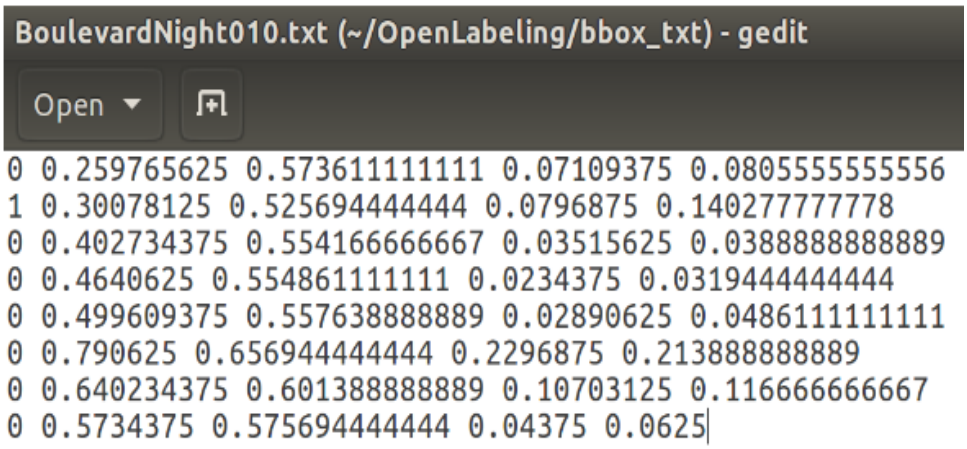
Automobil	11597
Kombi	1634
Autobus	2721
Kamion	1195

Za označavanje slika korišten je *OpenLabeling* [31] alat koji je javno dostupan na internetu, a koji je namijenjen korištenju na *Ubuntu* operacijskom sustavu. Ovaj alat omogućava označavanje objekata, promjenu veličine slike i detektiranje rubova. Navedeni alat moguće je instalirati naredbom u terminalu: *git clone* https://github.com/Cartucho/OpenLabeling. *OpenLabeling* alat za svoj rad zahtjeva instalaciju *Python* i *OpenCV* verzije 3 na računalo. Detaljne upute za instaliranje i pokretanje ovog alata mogu se naći na prethodno navedenom repozitoriju [31]. Nakon što se instalira *OpenLabeling*, bazu slika potrebno je prebaciti u mapu *Images* koja se kreira pri instalaciji ovog alata. U datoteci *class_list.txt* treba navesti sve klase objekata koje želimo koristiti, po jednu u svakom redu. Nakon što se obave sve potrebne promjene može se pokrenuti alat za označavanje naredbom u terminalu: *python main.py*. Nakon što se program pokrene, u mapi *bbox_txt* će se za svaku sliku iz mape *Images* kreirati po jedna tekstualna datoteka u koju će se prilikom označavanja slika spremati klasa označenog objekta i pozicija graničnog okvira. Slika 3.5. prikazuje izgled *OpenLabeling* alata. *OpenLabeling* alat se nalazi u prilogu P3.2. na priloženom DVDu.



Slika 3.5. *OpenLabeling alat*

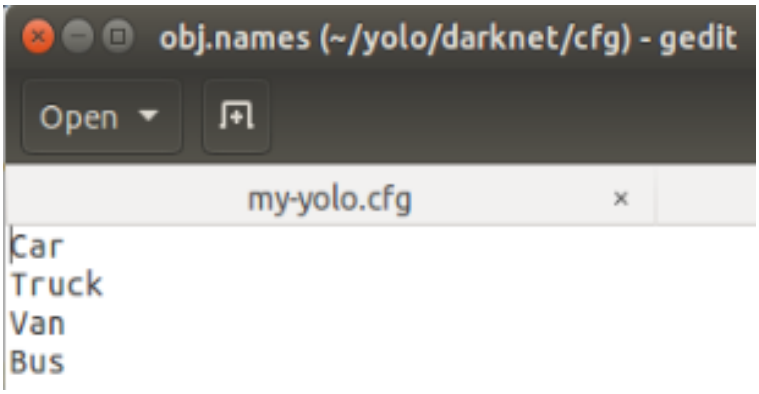
Na slici 3.6. je vidljiv izgled tekstualne datoteke s označenim objektima, te možemo vidjeti da je prva oznaka broj klase, druga i treća oznaka označavaju *x* i *y* koordinatu središta graničnog okvira dok ostale dvije oznake označavaju širinu i visinu graničnog okvira. Vrijednost visine i širine graničnog okvira je u rasponu od 0 i 1 zbog toga što ove vrijednosti označavaju relativnu vrijednost visine i širine objekta u odnosu na veličinu ulazne slike.



Slika 3.6. *Izgled jedne tekst datoteke nakon označavanja objekata na slici*

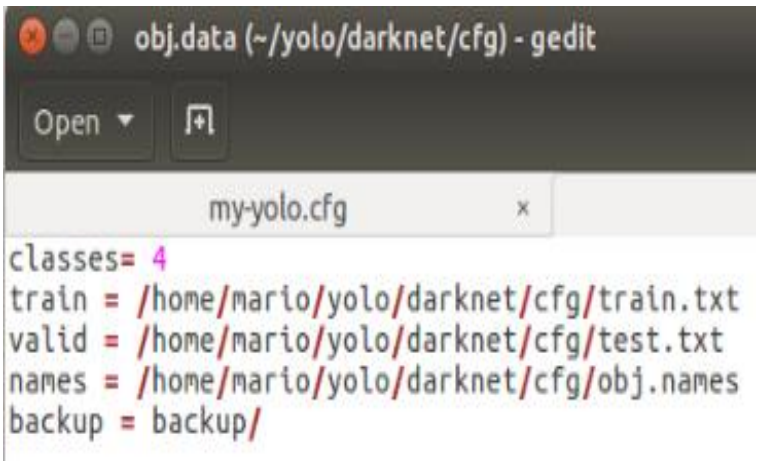
Nakon kreiranja nove baze slika za treniranje, potrebno je skinuti *Darknet* za *Yolo v2* algoritam naredbom u terminalu: *git clone* https://github.com/AlexeyAB/darknet.git. Nakon što se skine *Darknet*, u datoteci *Makefile* postaviti *GPU* s 0 na 1 da bi se za treniranje mreže koristila grafička kartica. Za treniranje neuronskih mreža preporučljivo je koristiti grafičke procesore s obzirom na to da se treniranje na njima brže obavlja nego na centralnim procesorima. Nakon promjene spremiti datoteku, te ponovno obaviti izgradnju pokretanjem naredbe *make* unutar *darknet* mape [32]. Da bi se mogao koristiti grafički procesor računala potrebno je imati grafički procesor koji podržava *CUDA Toolkit 9.0.*, te instalirati *CUDA*

Toolkit 9.0 s *Nvidia* stranice [33]. Nadalje, sve tekst i jpg datoteke trebaju biti spremljene u isti folder. Da bi Yolo v2 algoritam znao koje slike treba koristiti za treniranje, a koje za validaciju potrebno je kreirati dvije tekstualne datoteke koje će sadržavati putanje do slika za treniranje i validaciju. Za kreiranje *train.txt* i *test.txt* datoteka korištena je *Python* skripta *process.py* [31]. U *Python* skripti postaviti varijablu *percentage_test* na 10 što znači da će se 90% slika koristiti za treniranje, a 10% za validaciju. Skripta *process.py* se nalazi u prilogu P3.3 na priloženom DVDu. Nakon što se kreiraju *train* i *test* datoteke kreirati konfiguracijske datoteke. Prvo u mapi *cfg* kreirati dvije nove datoteke *obj.data* i *obj.names*. Datoteku *obj.names* popuniti nazivima klasa, u svakom redu po jedan naziv klase, ali istim redoslijedom kao u *OpenLabeling* alatu. Primjer izgleda *obj.names* datoteke na slici 3.7.



Slika 3.7. *Izgled obj.names datoteke*

U datoteku *obj.data* u prvom redu upisati broj klasa, u drugom redu putanju do *train.txt* datoteke, u trećem redu putanju do *test.txt* datoteke, u četvrtom redu putanju do *obj.names* datoteke, te u zadnji red upisati putanju na kojoj će se spremati vrijednost istreniranih parametara mreže. Izgled *obj.data* datoteke je vidljiv na slici 3.8.

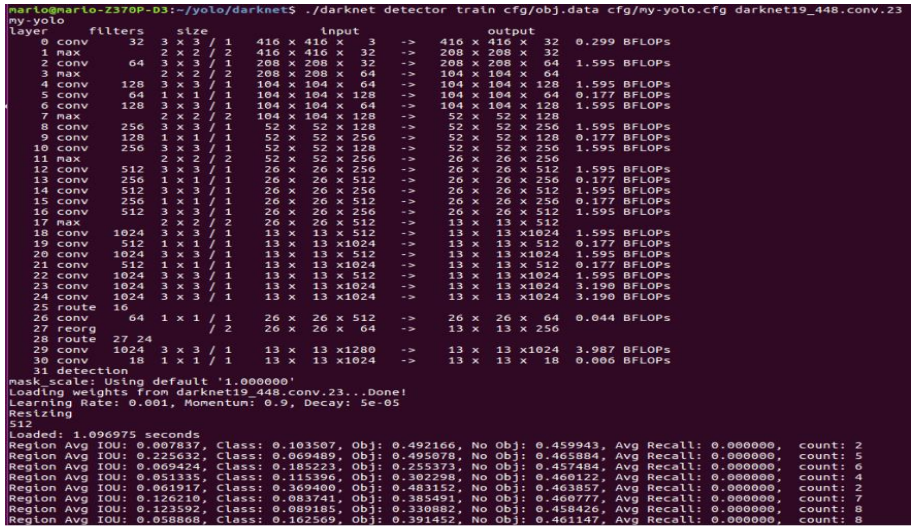


Slika 3.8. *Izgled obj.data datoteke*

Posljednja datoteka koju treba izmijeniti je *.cfg* datoteka. Prvo u *cfg* mapi treba napraviti duplikat *yolov2.cfg* datoteke. Na liniji 3 treba promijeniti *batch* u 64, a na liniji 4 *subdivision* u 32. Ove vrijednosti ovise o količini *VRAM* memorije koju posjeduje grafička kartica korištena za treniranje i stoga je ove vrijednosti moguće povećavati i smanjivati. Na liniji 17 se postavlja *learning_rate* koji je ostavljena na predefiniranoj vrijednosti 0.001. Nakon što se promijene parametri na liniji 120 varijablu *classes* postaviti u 4, jer želimo da mreža detektira 4 klase objekata. Varijablu *num* postaviti u 2, jer želimo da omogućimo da se u karti značajki koja je veličine 13 x 13 u jednom kvadratu mogu detektirati najviše 2 objekta. Obrisati višak vrijednosti u varijabli *anchors*, te ostaviti samo 4 prve vrijednosti. I na kraju promijeniti veličinu zadnjeg sloja u mreži. Za promjenu veličine zadnjeg sloja mreže na liniji 114 promijeniti vrijednost varijable *filters* u 18. Ovu vrijednost dobijemo korištenjem formule (3-2) [34]:

$$filters = (broj\ klasa + 5) * num \tag{3-2}$$

Broj *klasa* u ovom slučaju je 4, a vrijednost varijable *num* je 2. Nakon što su podešene sve konfiguracijske datoteke, potrebno je skinuti početne *Darknet* parametre mreže, koji će treniranjem biti prilagođeni za detekciju željenih klasa objekata. Parametri mreže se skidaju naredbom u terminalu: *wget* https://pjreddie/media/files/darknet53.conv.74. Nakon što su skinuti početni parametri mreže, može se pokrenuti treniranje konfigurirane *Yolo v2* mreže [34]. Konfiguracijske datoteke *Yolo v2* algoritma se nalaze u prilogu P3.4. na priloženom DVDu.Na slici 3.9. je prikazana naredba u terminalu za pokretanje treniranja i izgled terminala nakon pokretanja naredbe.

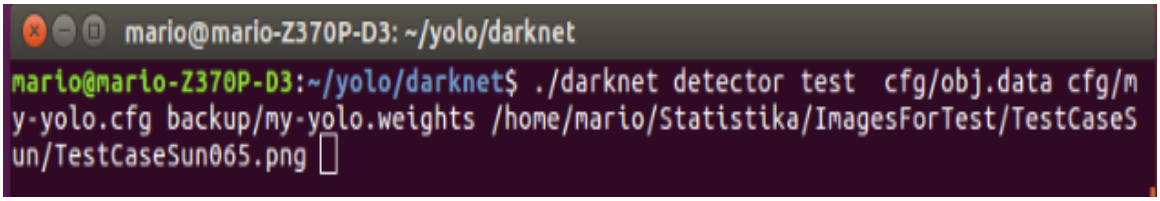


Slika 3.9. *Naredba za pokretanje treniranja*

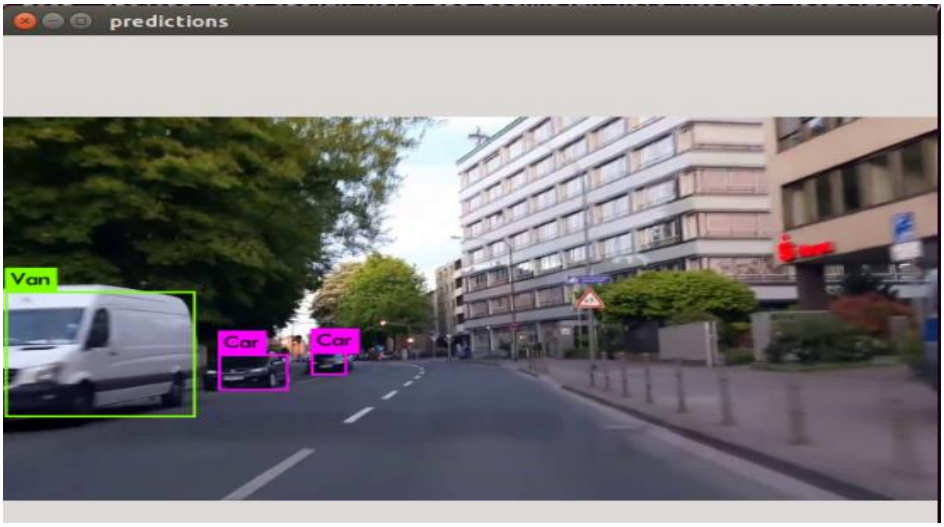
Minimalan broj epoha treniranja *Yolo v2* algoritma računa se prema formuli (3-3) [35]:

$$minimalan\ broj\ epoha = 2000 * broj\ klasa \tag{3-3}$$

Prema formuli (3-3) minimalan broj epoha treniranja *Yolo v2* algoritma je 8000, te je stoga *Yolo v2* algoritam treniran do 20 000 iteracija. Vrijednost srednjeg gubitka nakon treniranja je bio manji od 0.5, za što je bilo potrebno oko 26 sati treniranja. Nakon završetka treniranja kao rezultat se u mapi *backup* stvara datoteka *my-yolo.backup*, koja sadrži istrenirane parametre mreže za detektiranje vozila pomoću *Yolo v2* mreže. U *Darknetu* je moguće testirati detekciju vozila pomoću dobivenih parametara mreže. Slika 3.10. prikazuje naredbu za pokretanje detekcije, a slika 3.11. prikazuje rezultat detekcije.



Slika 3.10. *Naredba za pokretanje detekcije vozila na slici*



Slika 3.11. *Prikaz detektiranih vozila na slici*

3.3.2. Implementacija Yolo algoritma u ROS

Prije implementacije *Yolo v2* algoritma u *ROS* na računalu s *Ubuntu 16.04* operacijskim sustavom potrebno je instalirati *ROS Kinetic Kame*. Da bi se instalirao *ROS Kinetic Kame* na *ROS* stranici [36] treba odabrati *Ubuntu* platformu. Nakon toga treba pratiti upute za instalaciju, te instalirati *desktop* verziju *ROSa*. Uz *ROS* je potrebno instalirati i *OpenCV*

biblioteke. Detaljne upute za instalaciju *OpenCV* biblioteke mogu se naći na *codebind* stranici [37]. Nakon instalacije svih biblioteka treba kreirati novu radnu mapu na računalu. Da bi se kreirao novi *catkin wokrspace* u terminal unijeti sljedeću naredbu: *mkdir –p ~/catkin_ws/src*. Nakon što se mapa kreira, pozicionirati se u *catkin_ws* mapu i u terminal unijet sljedeću naredbu: *catkin build*.

Nakon kreiranja radne mape, da bi se implementirao *Yolo v2* algoritam u *ROS* s *GitHub* repozitorija *leggedrobotics / darknet_ros* treba skinuti *Darknet ROS* čvor. Prvo se treba pozicionirati u mapu *catkin_ws/src* i klonirati repozitorij sljedećom naredbom u terminalu: *git clone --recursive git@github.com: leggedrobotics/darknet_ros.git*. Nakon što se obavi kloniranje repozitorija pozicionirati se u mapu *catkin_ws* i izgraditi radnu mapu sljedećom naredbom: *catkin build darknet_ros –DCMAKE_BUILD_TYPE=Release*.

Da bi koristili novo istreniranu *Yolo v2* mrežu potrebno je kopirati konfiguracijske datoteke i težine u radnu mapu. U mapi na putanji *catkin_ws/src/darknet_ros/darknet_ros/yolo_network_cofig* se nalaze dvije mape *cfg* i *weights*. U mapu *cfg* kopirati *my-yolo.cfg* datoteku, a u mapu *weights* kopirati *my-yolo.weights* datoteku koju smo dobili treniranjem *Yolo v2* mreže. U mapi *catkin_ws/darknet_ros/darknet_ros/config* treba kreirati datoteku *my-yolo.yaml*. Datoteku popuniti kao na slici 3.12. U datoteci *my-yolo.yaml* postaviti ime konfiguracijske datoteke, ime datoteke koja sadrži težine, prag sigurnosti detekcije i nazive klasa objekata koji se žele detektirati istim redoslijedom kao što je to bio slučaj u *obj.names* datoteci koju smo koristili prilikom treniranja *Yolo v2* mreže.

```
yolo_model:

  config_file:
    name: my-yolo.cfg
  weight_file:
    name: my-yolo.weights
  threshold:
    value: 0.5
  detection_classes:
    names:
      - Car
      - Truck
      - Van
      - Bus
```

Slika 3.12. *Izgled my-yolo.yaml datoteke*

Nadalje, treba prepraviti datoteku *ros.yaml* da bi radila s novim konfiguracijskim datotekama. Primjer izgleda *ros.yaml* datoteke se nalazi na slici 3.13. Temu *camera_reading* postaviti u */videofile/image_raw* da bi čvor mogao koristiti video kao ulaz.

```
subscribers:
  camera_reading:
    topic: /videofile/image_raw
    queue_size: 1
actions:
  camera_reading:
    name: /darknet_ros/check_for_objects
publishers:
  object_detector:
    topic: /darknet_ros/found_object
    queue_size: 1
    latch: false
  bounding_boxes:
    topic: /darknet_ros/bounding_boxes
    queue_size: 1
    latch: false
  detection_image:
    topic: /darknet_ros/detection_image
    queue_size: 1
    latch: true
image_view:
  enable_opencv: true
  wait_key_delay: 1
  enable_console_output: true
```

Slika 3.13. *Izgled ros.yaml datoteke*

Nakon implementacije svih promjena ponovno izgraditi radnu mapu pomoću naredbe u terminalu: *catkin build*. Implementacija algoritma u *ROSu* se nalazi u prilogu P3.5. na priloženom DVDu.

3.3.3. Kreiranje čvora za procjenu udaljenosti

U ovom diplomskom radu razvijena su dva *ROS* čvora za procjenu udaljenosti. Prvi čvor za procjenu udaljenosti se koristi za procjenu udaljenosti u video snimkama iz stvarnog svijeta, dok se drugi čvor koristi za procjenu udaljenosti unutar *Carla* simulatora [38]. *Carla* simulator razvijen je za potporu razvoju, treningu i validaciji sustava za autonomnu vožnju. *Carla* simulator podržava razne vrste senzora, vremenskih uvjeta, te potpunu kontrolu statičkih i dinamičkih sudionika, generatora karti i dr. U ovom diplomskom radu korišten je *Carla* simulator verzija 0.9.6. Prije instalacije *Carla* simulatora treba instalirati i postaviti *Unreal Engine*. *Unreal Engine* je razvijen od strane *Epic Games* u *C++* programskom jeziku. Prije instalacije potrebno je registrirati se na *Unreal Engine* stranicu da bi dobili pristup repozitoriju na *GitHubu*. Nakon instalacije *Unreal Enginea*, instalirati *Carla* simulator i *Python API*, a detaljne upute za instalaciju se mogu naći na službenim stranicama *Carla* simulatora [39]. Slika 3.14. prikazuje izgled *Carla* simulatora.

Slika 3.14. *Izgled Carla simulatora* [39]

Razvijena su dva čvora zato što se rješenje za video snimke temelji na specifikaciji žarišne duljine leće kamere i širini senzora kamere kojom se video snima, dok u simulatoru ta dva parametra nisu specificirana. Oba navedena čvora koriste isti *Yolo v2* algoritam, koji prilikom detekcije objekata daje klasu objekta, koordinate graničnog okvira i vjerojatnost točne detekcije. S obzirom na to da je u jednoj slici moguće detektirati više vozila izmijenjena je poruka koju šalje *Yolo v2* algoritam, tako da sadrži i broj detektiranih objekata. Nakon što se objekt detektira, *Yolo v2* čvor šalje poruku ostalim čvorovima s parametrima detektiranih objekata.

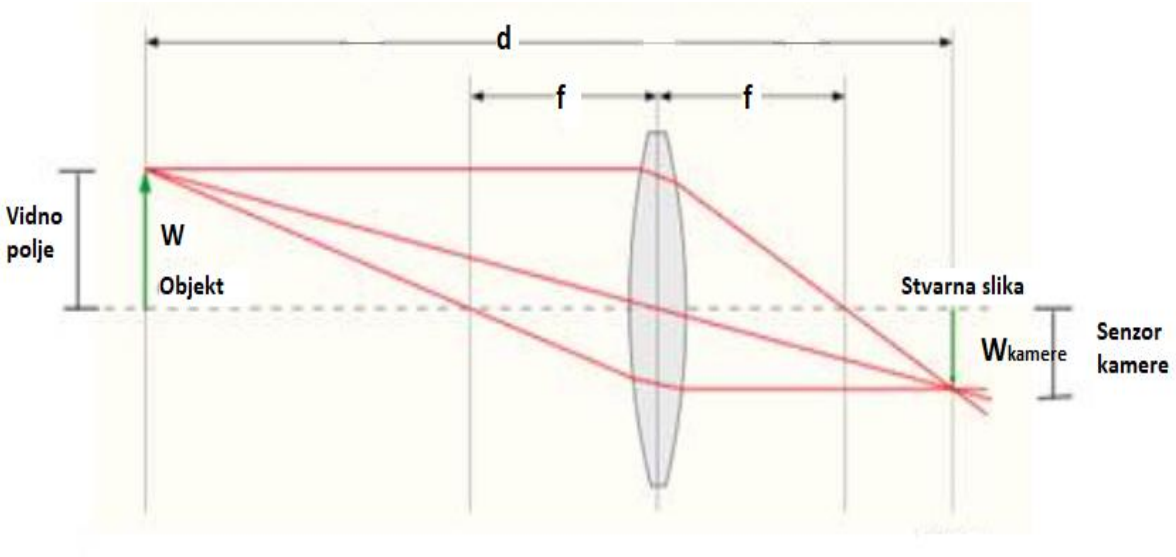
Prvi čvor je namijenjen korištenju na stvarnim video snimkama. Ovaj čvor, nakon što primi izlazne parametre iz *Yolo v2* algoritma, prvo na osnovu klase objekata određuje koja će se referentna širina vozila koristiti, jer svaka klasa vozila ima drugačiju širinu. Nakon toga se vrši procjena udaljenosti na temelju formule (3-4) [40]:

$$d[mm] = \frac{f[mm] \cdot w[mm] \cdot w_{slike}[element\ slike]}{w_{objekta}[element\ slike] \cdot w_{kamere}[mm]} \tag{3-4}$$

Gdje je:

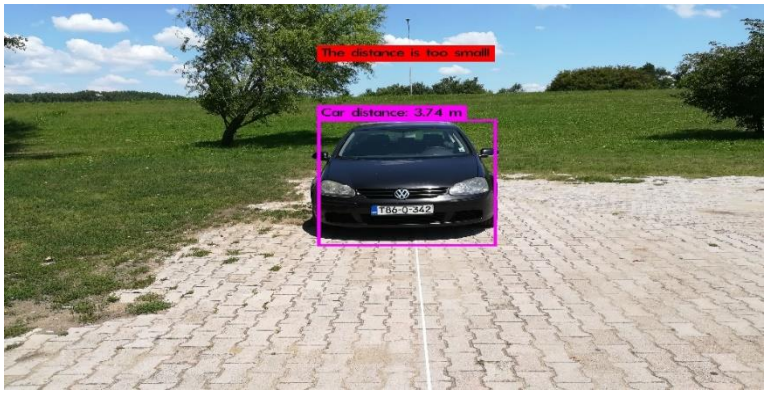
- d – udaljenost vozila od kamere u milimetrima
- f – žarišna duljina leće u milimetrima
- w – širina vozila u stvarnom svijetu u milimetrima
- w_{slike} – širina video okvira u elementima slike
- $w_{objekta}$ – širina objekta u video okviru u broju elemenata slike
- w_{kamere} – širina senzora kamere u milimetrima

Na slici 3.15. se nalazi fizikalni prikaz snimanja slike pomoću kamere na kojoj su prikazane osnovne veličine koje su korištene prilikom procjene udaljenosti na osnovu formule (3-4).

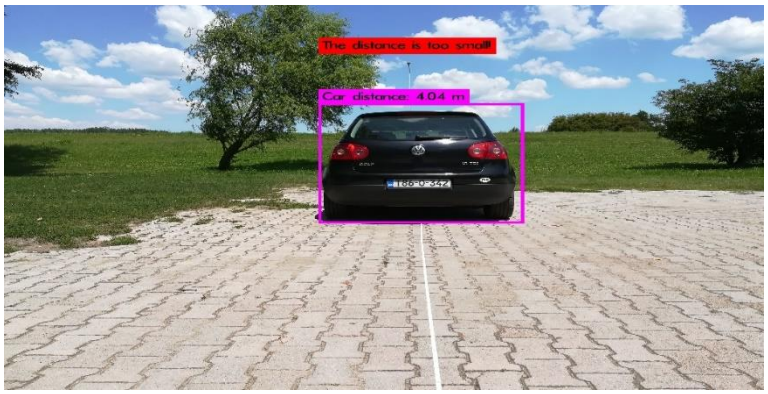


Slika 3.15. *Fizikalni prikaz snimanja slike pomoću kamere* [41]

Nakon procjene udaljenosti detektiranog vozila ispisuje se poruka upozorenja ako se detektirano vozilo nalazi preblizu, odnosno na udaljenosti manjoj od 5 metara. S obzirom na to da *Yolo v2* algoritam smanjuje sve slike na rezoluciju 416 x 416, širina video okvira u elementima slike je fiksna i iznosi 416. Širina vozila u stvarnom svijetu se razlikuje za svaku klasu vozila koje algoritam detektira. Prilikom odabira referentnih širina vozila, bilo je potrebno istražiti zakonske odredbe i prosječne širine vozila za svaku klasu vozila. Nakon istraživanja za referentnu širinu automobila odabrano je 1800 mm [42]. Za referentnu širinu kombija je odabrano 1900 mm [43]. Referentna širina koja je odabrana za autobus iznosi 2550 mm [44], a za kamion 2400 mm [45]. Iako su razlike u stvarnim širinama između automobila i kombija, te kamiona i autobusa jako male one donose veliku razliku prilikom procjene udaljenosti. Žarišna duljina leće i širina senzora kamere se određuje na osnovu kamere kojom je snimljen materijal za testiranje algoritma. Za testiranje algoritma u ovom diplomskom radu korištena je kamera mobilnog uređaja *Huawei P10 Lite*. Kamera mobilnog telefona *Huawei P10 Lite* kojom su snimane slike za testiranje algoritma ima širinu senzora od 6,7 mm [46], a žarišna duljina leće je 29 mm [47]. Slika 3.16. i 3.17. prikazuju primjer detekcije vozila i procjenu udaljenosti u stvarnom svijetu, te prikaz upozorenja.

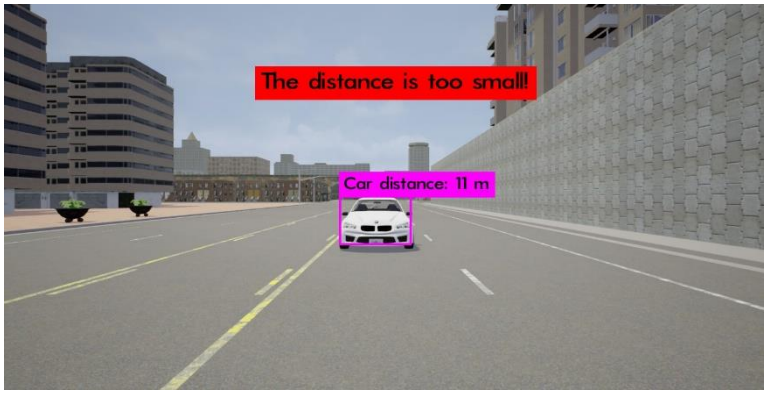


Slika 3.16. *Detekcija automobila i procjena udaljenosti za prednju stranu vozila u stvarnom svijetu na 5 m*

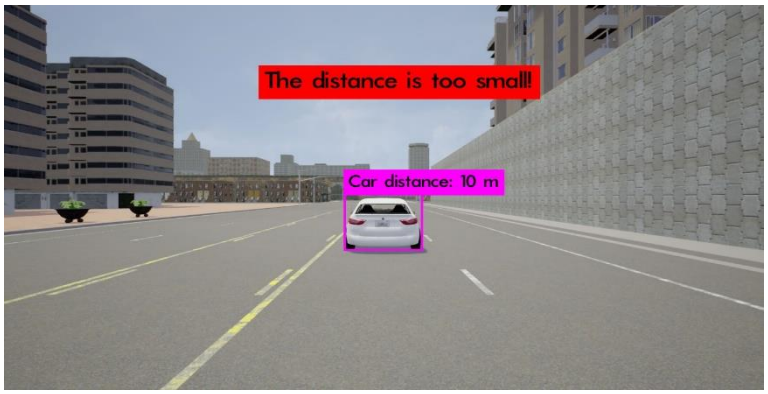


Slika 3.17. *Detekcija automobila i procjena udaljenosti za stražnju stranu vozila u stvarnom svijetu na 5 m*

Drugi čvor je razvijen zato što kamera u *Carla* simulatoru nema žarišnu duljinu leće i širinu senzora. Ovaj čvor je dobiven eksperimentalnim putem, tako što su uzeti uzorci (slike) vozila u simulatoru na udaljenostima od 5 do 50 metara, u koracima od 1 metar i gledana je širina vozila u elementima slike na svakoj od udaljenosti. Nakon proračuna kreiran je čvor koji pomoću širine vozila u elementima slike određuje udaljenost od kamere do detektiranog vozila. Za potrebe testiranja postavke za ispis upozorenja postavljene su tako, da se upozorenje prikazuje ako se vozilo nalazi na udaljenosti manjoj od 12 metara. Ovaj čvor kao takav se može samo primjenjivati u *Carla* simulatoru. Slika 3.18. i 3.19. prikazuju primjer detekcije vozila i procjene udaljenosti, te prikaza upozorenja u *Carla* simulatoru.



Slika 3.18. *Detekcija automobila i procjena udaljenosti za prednju stranu vozila u simulatoru na 10 m*



Slika 3.19. *Detekcija automobila i procjena udaljenosti za stražnju stranu vozila u simulatoru na 10 m*

4. EVALUACIJA RADA ALGORITMA ZA PROCJENU UDALJENOSTI VOZILA U OKOLINI AUTONOMNOG VOZILA

U ovom poglavlju prikazana je evaluacija rada algoritma za procjenu udaljenosti vozila u okolini autonomnog vozila. U potpoglavlju 4.1. opisano je okruženje na kojem je obavljano testiranje algoritma. U potpoglavlju 4.2. prikazani su rezultati testiranja *Yolo v2* algoritma koji je korišten za detekciju objekata u okolini autonomnog vozila. U potpoglavlju 4.3. prikazani su rezultati testiranja čvora za procjenu udaljenosti u stvarnom svijetu, dok su u potpoglavlju 4.4. prikazani rezultati testiranja čvora za procjenu udaljenosti u *Carla* simulatoru.

4.1. Okruženje za testiranje

Prilikom evaluacije rada algoritma za procjenu udaljenosti vozila u okolini autonomnog vozila korišteno je stolno računalo. Prilikom testiranja korišteno je izmijenjeno *Darknet* okruženje u koje su dodati čvorovi za procjenu udaljenosti u stvarnom svijetu i simulatoru. Specifikacije stolnog računala za testiranje algoritma prikazane su u tablici 4.1.

Tablica 4.1. *Prikaz specifikacija stolnog računala za obavljanje testiranja*

Procesor	Intel Core i5-8400
RAM	DDR4 16Gb Crucial 2400MHz Ballistix Sport LT
SSD	Kingston SSD A400
Grafička kartica	ASUS DUAL - GTX1060 - O3G (3Gb)
Matična ploča	Gigabyte GA-Z370P-D3
Operacijski sustav	Ubuntu 16.04

4.2. Testiranje Yolo v2 algoritma za detekciju objekata

U ovom potpoglavlju prikazani su rezultati testiranja *Yolo v2* algoritma za detekciju objekata. Prilikom testiranja algoritma, korišten je skup slika koji je dobiven iz video sekvenci po različitim vremenskim uvjetima. Ovaj skup slika je različit u odnosu na skup slika koji je korišten u procesu učenja. Skup slika za testiranje *Yolo v2* algoritma nalazi se u prilogu P4.1. na priloženom DVDu. Prilikom testiranja praćen je broj objekata u slikama, broj točnih

pozitivnih detekcija (engl. *True Positive - TP*), broj lažnih negativnih detekcija (engl. *False Negative - FN*) i broj lažnih pozitivnih detekcija (engl. *False Positive - FP*). U tablici 4.2. prikazani su rezultati testiranja *Yolo v2* algoritma za detekciju objekata u različitim vremenskim uvjetima. U tablici 4.2. je vidljivo da algoritam ima najviše problema pri detekciji objekata noću i po kiši. Razlog slabije detekcije u noćnim uvjetima vožnje dolazi zbog slabije osvijetljenosti, dok u kišnim uvjetima problem nastaje prilikom udaraca kapljica kiše po staklu ispred kamere što unosi veliku promjenu u samu snimku. Poboljšanje detekcije u ovakvim uvjetima moguće je proširenjem skupa za treniranje dodatnim slikama po noćnim i kišnim uvjetima vožnje.

Tablica 4.2. *Prikaz rezultata testiranja Yolo v2 algoritma za detekciju u različitim vremenskim uvjetima*

Testni slučaj	Vrsta objekta	Broj traženih objekata	TP	FN	FP
Magla	Automobil	81	58	23	2
	Kombi	68	62	6	0
	Kamion	0	0	0	0
	Autobus	0	0	0	0
Kiša	Automobil	211	178	33	0
	Kombi	16	9	7	0
	Kamion	11	8	3	0
	Autobus	4	4	0	2
Noć	Automobil	194	152	42	2
	Kombi	9	4	5	3
	Kamion	7	2	5	0
	Autobus	8	3	5	0
Sunce	Automobil	212	179	33	4
	Kombi	58	46	12	2
	Kamion	22	20	2	2
	Autobus	10	9	1	0

U tablici 4.3. prikazani su rezultati testiranja *Yolo v2* algoritma za detekciju po klasama. Kao što se može vidjeti u tablici 4.3. algoritam najbolje detektira automobile i kombije, dok ima slabiju detekciju kamiona i autobusa. Broj točnih detekcija klase kamion i autobus

moguće je povećati dodavanjem slika klase kamion i autobus u skup za treniranje *Yolo v2* algoritma i ponovnim treniranjem istog.

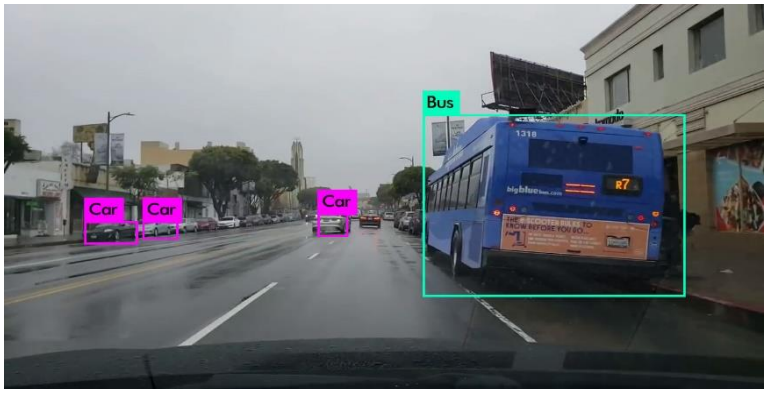
Tablica 4.3. *Prikaz rezultata testiranja Yolo v2 algoritma za detekciju po klasama objekta*

Klasa objekta	Broj traženih objekata	TP	FN	FP
Automobil	698	567	131	8
Kombi	151	121	30	5
Kamion	40	30	10	2
Autobus	22	16	6	2

Slika 4.1. prikazuje točnu detekciju automobila po noći. Slika 4.2. prikazuje detekciju automobila i kombija u maglovitim uvjetima vožnje. Slika 4.3. prikazuje detekcije automobila i autobusa u kišnim uvjetima na cesti. Slika 4.4. prikazuje točnu detekciju kamiona u sunčanim uvjetima vožnje. Slika 4.5. prikazuje primjer pogrešne detekcije automobila kao kombija. Slika 4.6. prikazuje primjer detekcije više vozila koji se preklapaju s tim da je jedno od vozila pogrešno detektirano, kombi vozilo je detektirano kao automobil. Slika 4.7. prikazuje detekciju više vozila na slici u kojoj su vozila samo djelomično vidljiva.

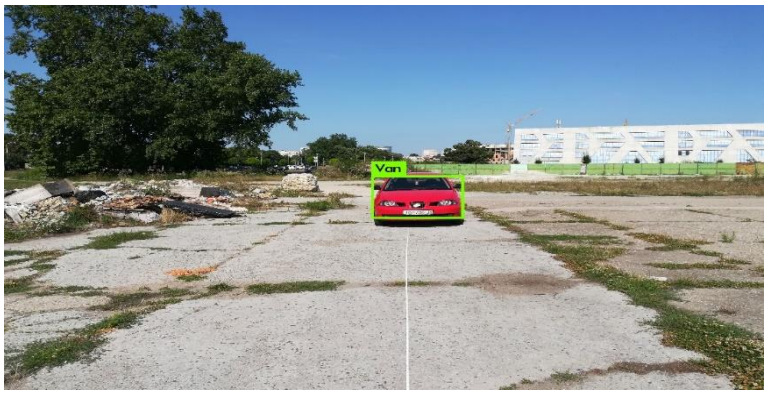
Slika 4.1. *Detekcija objekata pomoću Yolo v2 algoritma po noći*

Slika 4.2. *Detekcija objekata pomoću Yolo v2 algoritma po magli*

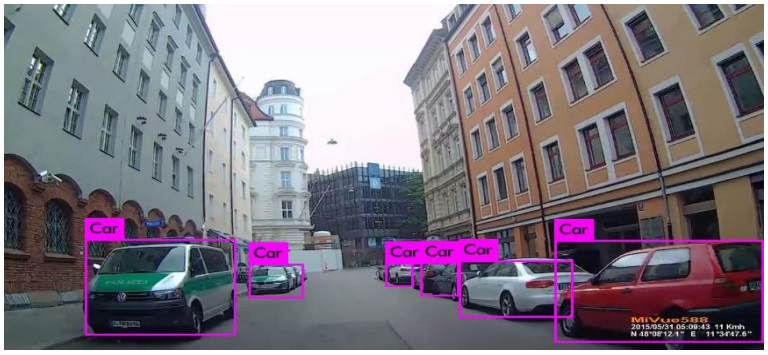


Slika 4.3. *Detekcija objekata pomoću Yolo v2 algoritma po kiši*

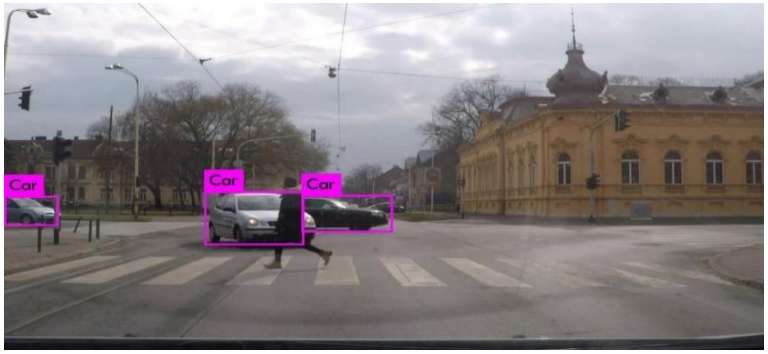
Slika 4.4. *Detekcija objekata pomoću Yolo v2 algoritma po danu*



Slika 4.5. *Primjer pogrešne detekcije automobila kao kombija*



Slika 4.6. *Primjer pogrešne detekcije kombija kao automobila*



Slika 4.7. *Primjer detekcije automobila*

4.3. Testiranje čvora za procjenu udaljenosti u stvarnom svijetu

U ovom potpoglavlju prikazana je evaluacija čvora za procjenu udaljenosti u stvarnom svijetu. Evaluacija je obavljena na slikama četiri klase objekata na odaljenosti od 5 do 45 metara. Testiranje je obavljeno na udaljenosti do 45 metara, jer je tijekom testiranja utvrđeno da *Yolo v2* algoritam ne detektira objekte koji su udaljeniji od 45 metara. Baza slika za testiranje čvora za procjenu udaljenosti u stvarnom svijetu nalazi se u prilogu P4.2. na priloženom DVDu. Razlog ne detektiranja objekata koji su udaljeniji od 45 metara je što se *Yolo v2* algoritam trenirao na slikama veličine 416 x 416. Ovaj problem bi se mogao riješiti treniranjem *Yolo v2* algoritma na većim slikama, pa bi algoritam bolje detektirao manje objekte u slici. U tablici 4.4. prikazani su rezultati procjene udaljenosti za detektirane automobile. Iz tablice 4.4. se može vidjeti da povećanjem udaljenosti detektiranog objekta od kamere vozila, povećava se i odstupanje srednje vrijednosti od stvarne vrijednosti udaljenosti detektiranog vozila, a to se događa zato što se na većim udaljenostima veličina graničnog okvira u elementima slike sve manje razlikuje s obzirom na to da je slika veličine 416 x 416. Kada bi se povećala veličina slike razlika veličine graničnog okvira u elementima slike na većim udaljenostima bi se više razlikovala što bi povećalo preciznost izračuna udaljenosti. Također, granični okvir pri manjim udaljenostima ima manje odstupanje od graničnog okvira

na većim udaljenostima. Odstupanje graničnog okvira je manje na manjim udaljenostima jer je objekt veći i detektor ga može bolje detektirati, dok na većim udaljenostima to odstupanje postaje veće jer je objekt sve manji i teže ga je precizno izdvojiti iz okoline. Također, prilikom testiranja na testnom skupu slika je određena vrijednosti *IoU*, koja govori koliko se prosječno granični okvir detekcije poklapa sa stvarnim graničnim okvirom koji je označen tijekom kreiranja skupa za testiranje. *IoU* za testni skup slika iznosi 64.24 %. Sve testne slike snimljene su kamerom mobilnog uređaja *Huawei P10 Lite*.

Tablica 4.4. *Rezultati procjene udaljenosti za klasu Automobil u stvarnom svijetu*

		Procijenjena udaljenost na slikam iz stvarnog svijeta	
Udaljenost	Objekt	Srednja vrijednost	Standardna devijacija
5	Automobil	4,48	0,22198
10		8,69	0,31469
15		13,82	0,57518
20		18,07	1,15181
25		22,93	0,91596
30		26,72	1,19939
35		32,34	1,97104
40		36,21	2,11445
45		40,57	2,97864

a)

b)

c)

d)


Car distance: 8.54 m

e)


Car distance: 8.19 m

f)


Car distance: 8.5 m

g)


Car distance: 9.31 m

h)


Car distance: 8.62 m

i)


Car distance: 9.21 m

j)


Car distance: 8.58 m

k)


Car distance: 8.48 m

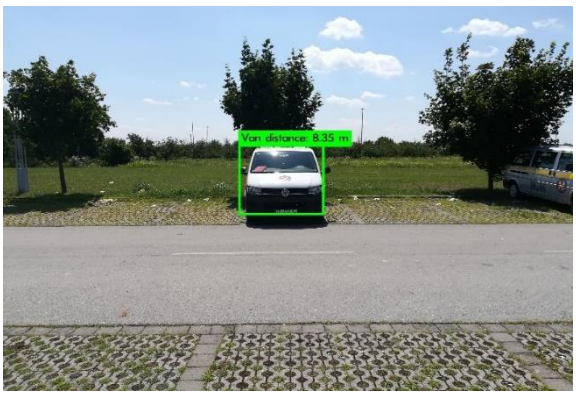
l)

m)

Slika 4.8. *Prikaz detekcije i procjene udaljenosti za prednju stranu a), c), e), g), i), k) i stražnju stranu b), d), f), h), j), l), m) automobila na 10 m*

Tablica 4.5. *Rezultati procjene udaljenosti za klasu Kombi u stvarnom svijetu*

		Procijenjena udaljenost na slikam iz stvarnog svijeta	
Udaljenost	Objekt	Srednja vrijednost	Standardna devijacija
5	Kombi	4,27	0,27114
10		8,49	0,54017
15		13,08	0,70378
20		18,08	1,02369
25		22,65	1,59276
30		26,76	1,62309
35		32,03	1,71631
40		36,24	2,63191
45		39,08	2,99355



a)

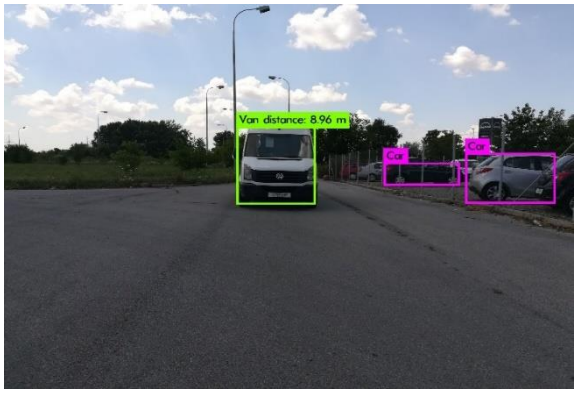
b)


Van distance: 8.22 m

c)


Van distance: 8.97 m

d)


Van distance: 8.96 m
Car
Car

e)


Van distance: 8.38 m
Car
Car

f)


Van distance: 7.65 m

g)


Van distance: 8.29 m

h)


Van distance: 8.65 m

i)


Van distance: 8.6 m

j)

k)

Slika 4.9. *Prikaz detekcije i procjene udaljenosti za prednju stranu a), c), e), g), i), k) i stražnju stranu b), d), f), h), j) kombija na 10 m*

Tablica 4.6. *Rezultati procjene udaljenosti za klasu Kamion u stvarnom svijetu*

		Procijenjena udaljenost na slikam iz stvarnog svijeta	
Udaljenost	Objekt	Srednja vrijednost	Standardna devijacija
5	Kamion	4,23	0,14923
10		7,96	0,39896
15		13,98	1,12460
20		17,58	1,13700
25		21,38	1,57217
30		25,59	1,29382
35		32,27	2,02131
40		35,87	2,31184
45		39,67	2,85619

a)

b)

c)

d)

e)

f)

Slika 4.10. *Prikaz detekcije i procjene udaljenosti za prednju stranu a), c) i stražnju stranu b), d), e), f) kamiona na 10 m*

Tablica 4.6. *Rezultati procjene udaljenosti za klasu Autobus u stvarnom svijetu*

		Procijenjena udaljenost na slikam iz stvarnog svijeta	
Udaljenost	Objekt	Srednja vrijednost	Standardna devijacija
5	Autobus	4,4	0,21195
10		8,42	0,50245
15		13,66	1,11342
20		17,11	1,30925
25		22,32	1,65886
30		27,41	2,42370
35		31,61	2,92128
40		35,19	3,17387
45		39,74	3,49055

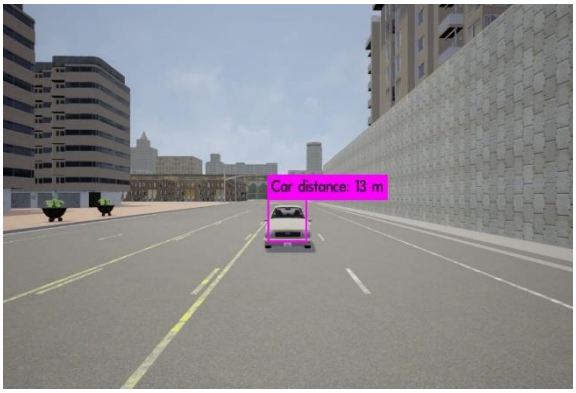
Slika 4.11. *Prikaz detekcije i procjene udaljenosti za prednju stranu d), f) i stražnju stranu a), b), c), e), g), h) autobusa na 10 m*

4.4. Testiranje čvora za procjenu udaljenosti u simulatoru

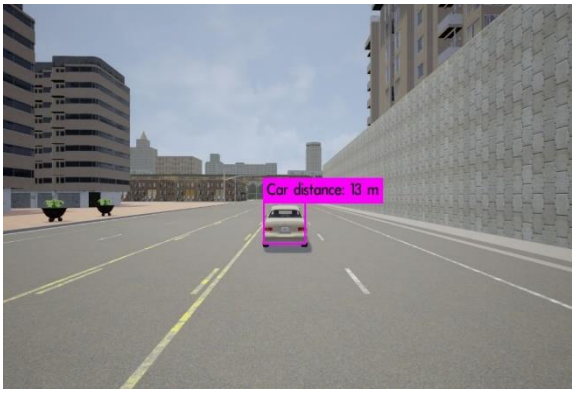
U ovom potpoglavlju prikazana je evaluacija čvora za procjenu udaljenosti u simulatoru. Evaluacija je obavljena na 10 skupova slika koje se sastoje od 5 različitih automobila, s prednje i zadnje strane na udaljenosti od 10 do 50 metara, jer je korištena osnovna verzija *Carla* simulatora koji ima ograničen broj vozila, te je svako dodatno vozilo u simulatoru je potrebno kupiti. Testiranje je obavljeno na udaljenosti do 50 metara, jer je tijekom testiranja utvrđeno da *Yolo v2* algoritam ne detektira objekte koji su udaljeniji od 50 metara. Baza slika za testiranje čvora za procjenu udaljenosti u simulatoru nalazi se u prilogu P4.3. na priloženom DVDu. Jedan od razloga ne detektiranja objekata je taj što u simulatoru objekti nakon 50 metara postaju jako mali i mutni. Drugi razlog ne detektiranja objekata koji su udaljeniji od 50 metara je što se *Yolo v2* algoritam trenirao na slikama veličine 416 x 416. Ovaj problem bi se mogao riješiti treniranjem *Yolo v2* algoritma na većim slikama, pa bi algoritam bolje detektirao manje objekte u slici. U tablici 4.8. prikazani su rezultati procjene udaljenosti. Kao što se u tablici 4.8. može vidjeti, srednja vrijednost procijenjene udaljenosti odstupa od točne vrijednosti udaljenosti u simulatoru. Također se može vidjeti, da se standardna devijacija povećava s povećanjem udaljenosti detektiranog objekta od kamere vozila. Također, prilikom testiranja na testnom skupu slika je određena vrijednosti *IoU,* koji govori koliko se prosječno granični okvir detekcije poklapa sa stvarnim graničnim okvirom koji je označen tijekom kreiranja skupa za testiranje. *IoU* za testni skup slika iznosi 59.48 %.

Tablica 4.8. *Rezultat procjene udaljenosti za klasu Automobil u simulatoru*

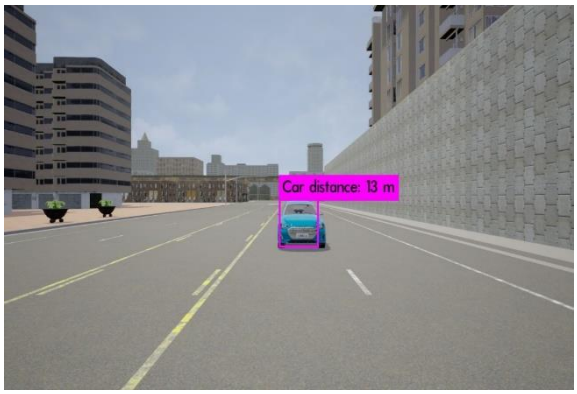
		Procijenjena udaljenost na slikama iz Carla simulatora	
Udaljenost	Objekt	Srednja vrijednost	Standardna devijacija
10	Automobil	12,125	1,02469
15		16,0625	1,91376
20		21,0625	1,98221
25		25,1875	1,90503
30		29,875	2,33452
35		34,5	2,70801
40		39,5	2,92118
45		44,0625	3,45386
50		49,125	2,33458


Car distance: 13 m

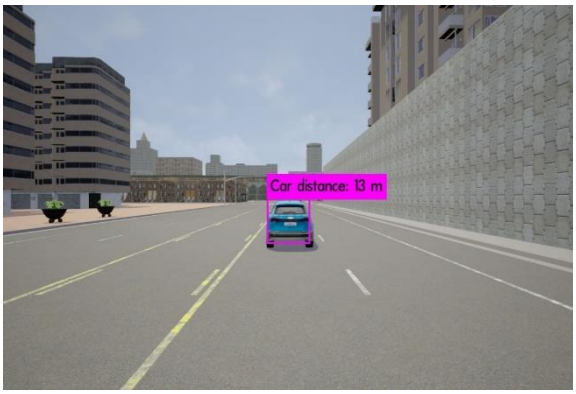
a)


Car distance: 13 m

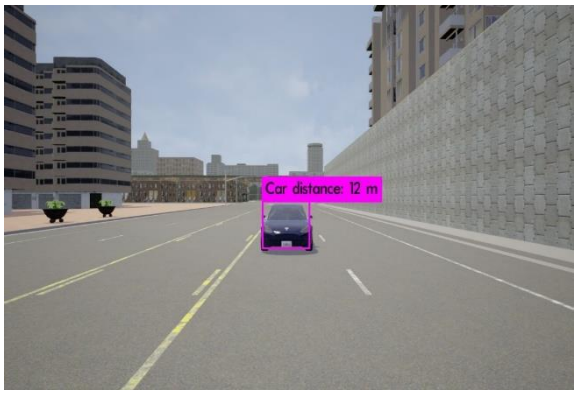
b)


Car distance: 13 m

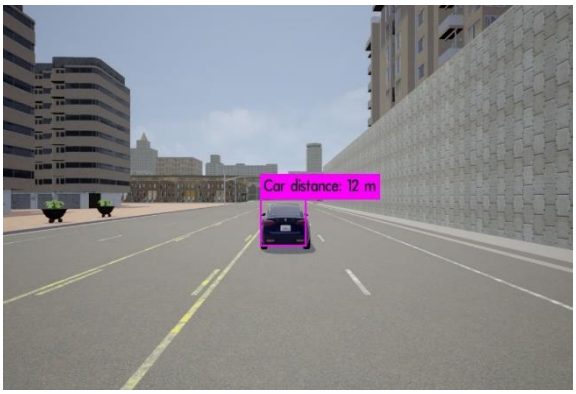
c)


Car distance: 13 m

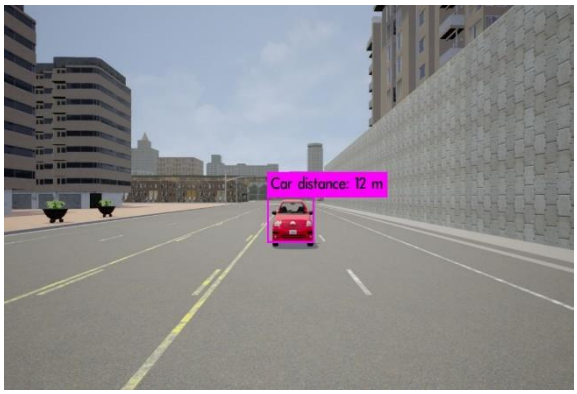
d)


Car distance: 12 m

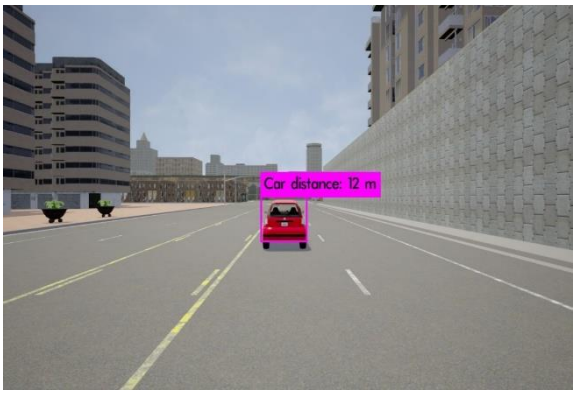
e)


Car distance: 12 m

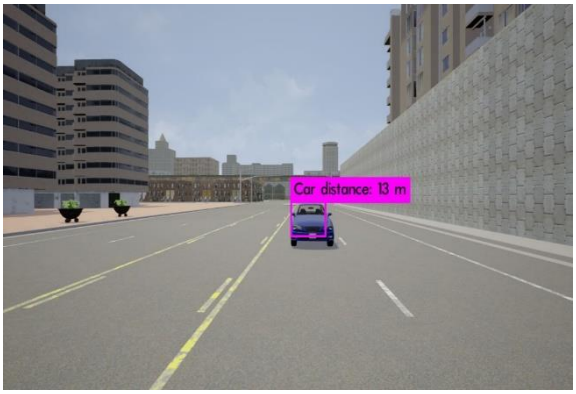
f)


Car distance: 12 m

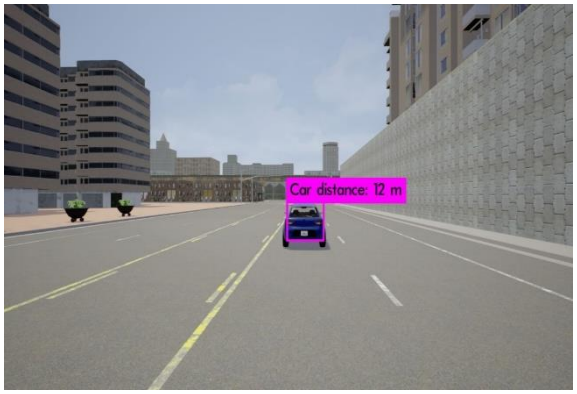
g)


Car distance: 12 m

h)

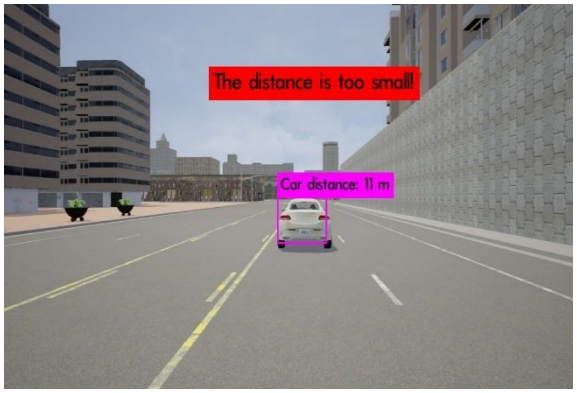


i)

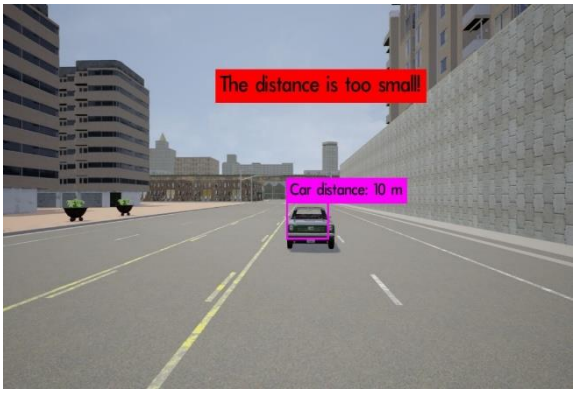


j)

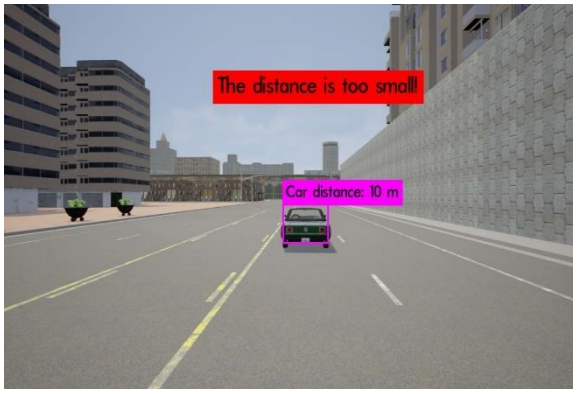
k)



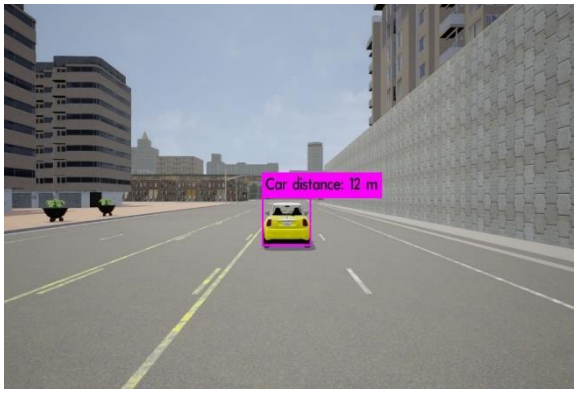
l)



m)



n)



o)

Slika 4.12. *Prikaz detekcije i procjene udaljenosti za prednju stranu a), b), e), g), i), k), m) i stražnju stranu b), d), f), h), j), l), n), o) automobila 10 m*

5. ZAKLJUČAK

S obzirom na to da se prometne nesreće događaju svakodnevno, velika potreba je za sustavom, koji može detektirati vozila u okruženju i procijeniti hoće li doći do potencijalne kolizije s vozilom. Ovakvi sustavi pomažu vozaču pri vožnji i povećavaju sigurnost putnika u prometu. U ovom diplomskom radu, prikazano je rješenje za detekciju vozila u okolini autonomnog vozila i upozorenja na potencijalnu koliziju. Sustav je implementiran u *ROSu* te se sastoji iz dva dijela. Prvi dio sustava koristi se za detekciju vozila u okruženju autonomnog vozila. Dio za detekciju sastoji se od *Yolo v2* algoritma, treniranog na novo kreiranom skupu slika. *Yolo v2* algoritam konfiguriran je tako da detektira četiri klase objekata: automobil, kombi, kamion i autobus. Detekcije su razdvojene u četiri klase vozila, zato što se ova vozila razlikuju po svojim dimenzijama. Drugi dio sustava je *ROS* čvor za procjenu udaljenosti. U ovom diplomskom radu izrađena su dva *ROS* čvora za procjenu udaljenosti, s tim da se jedan *ROS* čvor koristi za procjenu udaljenosti u *Carla* simulatoru, s kojim je povezan pomoću *ros_bridge*, dok se drugi *ROS* čvor koristi za procjenu udaljenosti u stvarnom svijetu. Procjena udaljenosti u *Carla* simulatoru, obavlja se pomoću funkcije koja za određenu širinu vozila u elementima slike kao izlaz daje udaljenost od kamere do vozila. *ROS* čvor za procjenu udaljenosti u stvarnom svijetu zasnovan je na specifikacijama kamere, referentnoj širini vozila u stvarnom svijetu i širini graničnog okvira detektiranog objekta.

Provedenim testiranjem sustava za detekciju vozila u okolini autonomnog vozila i upozorenja na potencijalnu koliziju, uočeno je da algoritam radi na prosječno 40 okvira u sekundi na *Nvidia GTX1060 (3Gb)* grafičkoj kartici, što znači da algoritam radi u stvarnom vremenu. Također, za razliku od drugih detektora kao što je *Viola-Jones*, razvijeni algoritam omogućava detekciju objekata i kada se samo dio objekta nalazi unutar slike ili kada se više objekata preklapa. Moguća je detekcija vozila iz svih pozicija oko vozila. Na slici 4.6. vidljiva je detekcija više automobila koji se preklapaju, dok je na slici 4.7. vidljiva detekcija automobila koji je samo djelomično vidljiv na slici. Razvijeni algoritam je lako prilagodljiv, te ga je lako nadograditi da detektira i druge objekte u prometu. Kao što se iz provedenih testova može vidjeti algoritam vrlo dobro obavlja procjenu udaljenosti objekta na manjim udaljenostima, te se sve većim udaljavanjem objekta greška u procjeni povećava. Razvojem i testiranjem algoritma uočeno je da algoritam ima slabiju detekciju vozila po noći i kiši. Također, uočeno je da sustav slabije detektira objekte klase kamion i autobus. Prilikom testiranja algoritma na slikama s objektima na određenim udaljenostima, uočeno je da algoritam prilikom udaljavanja objekta od kamere sve lošije označava granične okvire oko

detektiranih objekata, te da na udaljenostima preko 50 metara algoritam ne detektira objekte. Nedostatke problema lošije detekcije po noći i kiši, te slabije detekcije objekata klase kamion i autobus moguće je poboljšati treniranjem *Yolo v2* algoritma na većem skupu slika kojem bi se dodale slike kamiona i autobusa, te ostalih objekata u noćnim i kišnim slikama. Problem ne detektiranja objekata na udaljenostima većim do 50 metara moguće je riješiti treniranjem *Yolo v2* algoritma na slikama veće rezolucije. Prilikom testiranja procjene udaljenosti uočeno je da povećanjem udaljenosti vozila od kamere povećava se pogreška procjene, jer na većim udaljenostima granični okvir ima lošiju detekciju što se može riješiti treniranjem algoritma na slikama veće rezolucije. Pogrešku u procjenu udaljenosti unosi i položaj kamere u odnosu na vozilo, kvaliteta kamera koja se koristi za snimanje, te loše definirani parametri kamere. Neke od ovih problema još uvijek nije moguće izbjeći, stoga procjena udaljenosti neće biti uvijek točna.

LITERATURA

[1] P. YoonSeok, C. HanCheol, J. RyuWoon, L. TaeHoon, ROS Robot Programming 2017., ROBOTIS CO.

[2] YOLO: Real-Time Object Detection, https://pjreddie.com/darknet/yolov2/ [28. lipanj 2019.]

[3] K. Giseok, C. Jae-Soo, Vision-based vehicle detection and inter-vehicle distance estimation

[4] W. Berger, Deep Learning Haar Cascade Explained, http: www.willberger.org/cascade-haar-explained/ [28. lipanj 2019.]

[5] A. Rosebrock, Sliding Windows for Object Detection with Python and OpenCV, https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/ [28. lipanj 2019.]

[6] T. Babb, How a Kalman filter works in pictures, https: www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/ [28. lipanj 2019.]

[7] C. Chao-Ho, C. Tsong-Yi, H. Deng-Yuan, F. Kai-Wei, Front vehicle detection and distance estimation using single-lens video camera

[8] Vanishing points and horizons, Applications of projective transformations, https://cs.adelaide.edu.au/users/ianr/Teaching/CompGeom/lec2.pdf [28. lipanj 2019.]

[9] J. Liang, Canny Edge Detection, http://justin-liang.com/tutorials/canny/ [28. lipanj 2019.]

[10] R. Fisher, S. Perkins, A. Walker, E. Wolfart, Hough Transform, http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm [28. lipanj 2019.]

[11] A. Fagna, Understanding Edge Detection (Sobel Operator), https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900 [28. lipanj 2019.]

[12] G. Xiao-Feng, C. Zi-Wei, M. Ting-Song, L. Fan, Y. Long, Real-time vehicle detection and tracking using deep neural networks

[13] T. Sik-Ho, GoogLeNet (Inception v1) – Winner of ILSVRC 2014 (Image Classification), https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7 [28. lipanj 2019.]

[14] S. Das, CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more, https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5 [30. lipanj 2019.]

[15] 7 Types of Neural Network Activation Functions: How to Choose?, https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/ [30. lipanj 2019.]
[16] A. Abduladhem Abdulkareem, H. Hussein Alaa, Distance estimation and vehicle position detection based on monocular camera
[17] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, 2001, pp. I-511-I-518 vol. 1.
[18] OpenCV official website, https://opencv.org/ [1. srpanj 2019.]
[19] R. Gandhi, R-CNN, Fast R-CNN, Faster R-CNN, Yolo – Object Detection Algorithms, https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e [1. srpanj 2019.]
[20] J. Redmon, S. Divvala, R. Girshick, A. Farahadi, You Only Look Once: Unified, Real-Time Object Detection, University of Washington, Allen Institute for AI, Facebook AI Research
[21] A. Rosebrock, Intersection over Union (IoU) for object detection, https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ [1. srpanj 2019.]
[22] J. Hui, Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3, https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088 [1. srpanj 2019.]
[23] V. K. Jonnalagadda, Object Detection YOLO v1, v2, v3, https://medium.com/@venkatakrishna.jonnalagadda/object-detection-yolo-v1-v2-v3-c3d5eca2312a [1. srpanj 2019.]
[24] M. Pgliese, A very shallow overview of YOLO and Darknet, https://martinapugliese.github.io/recognise-objects-yolo/ [1. srpanj 2019.]
[25] C Language Introduction, https://www.geeksforgeeks.org/c-language-set-1-introduction/ [1. srpanj 2019.]
[26] CUDA Zone official website, https://developer.nvidia.com/cuda-zone [2. srpanj 2019.]
[27] J. Hui, Understanding Feature Pyramid Networks for object detection (FPN), https://medium.com/@jonathan_hui/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c [2. srpanj 2019.]
[28] ImageNet baza slika, http://www.image-net.org/ [2. srpanj 2019.]
[29] PascalVOC baza slika, http://host.robots.ox.ac.uk/pascal/VOC/ [2. srpanj 2019.]

[30] KITTI baza slika, http://www.cvlibs.net/datasets/kitti/ [2. srpanj 2019.]

[31] J. Cartucho, OpenLabeling: open-source image and video labeler, https://github.com/Cartucho/OpenLabeling [2. srpanj 2019.]

[32] M. Murugavel, How to train YOLOv2 to detect custom objects, https://medium.com/@manivannan_data/how-to-train-yolov2-to-detect-custom-objects-9010df784f36 [2. srpanj 2019.]

[33] CUDA Toolkit 9.0 official website, https://developer.nvidia.com/cuda-90-download-archive [4. srpanj 2019.]

[34] Darknet for YOLO, https://pjreddie.com/darknet/yolo/ [4. srpanj 2019.]

[35] Yolo-v3 and Yolo-v2 for Windows and Linux, https://github.com/AlexeyAB/darknet [5. srpanj 2019.]

[36] ROS official website, https://www.ros.org/ [5. srpanj 2019.]

[37] How to Install OpenCV in Ubuntu 18.04 LTS for C/C++, http://www.codebind.com/linux-tutorials/install-opencv-ubuntu-18-04-lts-c-cpp- [5. srpanj 2019.]

[38] Carla simulator official website, http://carla.org/ [5. srpanj 2019.]

[39] How to build CARLA on Linux, https://carla.readthedocs.io/en/latest/how_to_build_on_linux/ [5. srpanj 2019.]

[40] How do I calculate the distance of an object in a photo?, https://photo.stackexchange.com/questions/12434/how-do-i-calculate-the-distance-of-an-object-in-a-photo [6. srpanj 2019.]

[41] Physics Behind the Camera Lens, http://www.odec.ca/projects/2007/aust7k2/Principles.htm [26. kolovoz 2019.]

[42] Understanding Car Size and Dimensions, https://www.nationwidevehiclecontracts.co.uk/blog/understanding-car-size-and-dimensions [6. srpanj 2019.]

[43] Van dimensions and comparisons, https://sportsmobile.com/van-dimensions/ [25. lipanj 2019.]

[44] Prometna zona, Portal za promet i prometnu znanost, https://www.prometna-zona.com/autobusi/ [25. lipanj 2019.]

[45] Which Is The Best Truck Size For You, https://kargo.tech/artikel/which-is-the-best-truck-size-for-you/ [25. lipanj 2019.]

[46] Širina senzora kamere Sony IMX286, https://en.wikipedia.org/wiki/Exmor [25. lipanj 2019.]

[47] Huawei P10 Lite Review, https://www.photographyblog.com/reviews/huawei_p10_lite_review/image_quality [25. lipanj 2019.]

SAŽETAK

Tema ovog diplomskog rada je „Detekcija vozila u okruženju autonomnog vozila u svrhu upozorenja na potencijalnu koliziju“. Jedna od važnih stvari u autonomnoj vožnji je percepcija okoline vozila. U ovom radu razvijen je sustav koji detektira vozila u okruženju autonomnog vozila i obavlja procjenu udaljenosti detektiranih vozila. Za detekciju vozila koristi se *Yolo v2* algoritam koji je istreniran na novo kreiranom skupu slika, te obavlja detekciju četiri klase vozila: automobil, kombi, kamion i autobus. Sustav je implementiran u *ROS* pomoću čvorova. Za procjenu udaljenosti kreirana su dva čvora. Jedan čvor se koristi za procjenu udaljenosti u *Carla* simulatoru i dobiven je eksperimentalnim putem, dok se drugi čvor koristi za procjenu udaljenosti u stvarnom svijetu i zasnovan je na procjeni udaljenosti pomoću parametara kamere i referentne širine vozila u stvarnom svijetu. Obavljeno je testiranje algoritma na stolnom računalu, korištenjem slika iz simulatora i stvarnog svijeta. Uočeno je, da algoritam radi na prosječno 40 okvira u sekundi na *Nvidia GTX1060 (3Gb)* grafičkoj kartici, što znači da algoritam radi u stvarnom vremenu. Razvijeni algoritam omogućava detekciju objekata i kada se samo dio objekta nalazi unutar slike ili kada se više objekata preklapa. Moguća je detekcija vozila iz svih pozicija oko vozila. Prilikom testiranja je uočeno, da algoritam ne detektira objekte na udaljenosti većoj od 50 metara, te da slabije detektira objekte po noći i kiši. Također je uočeno, da se povećavanjem udaljenosti preciznost procijenjene udaljenosti smanjuje, te da je potrebno obaviti određene dorade da bi sustav imao bolju detekciju i preciznost procjene udaljenosti. Razvijeni algoritam je lako prilagodljiv, te ga je lako nadograditi da detektira i druge objekte u prometu.

Ključne riječi: *detekcija objekata, ROS, Yolo, Yolo v2, procjena udaljenosti, Carla simulator, kolizija, autonomno vozilo*

ABSTRACT

The topic of this masterwork is "Detection of vehicles in an autonomous vehicle environment for the purpose of warning of potential collision". One of the important things in an autonomous drive is the perception of the vehicle's environment. In this paper is developing a system that detects vehicles in an autonomous vehicle environment and carries out an assessment of the distance of detected vehicles. For vehicle detection, a Yolo v2 algorithm is used, which is trained on a newly created image set and carries out detection of four classes of vehicle: car, van, truck and bus. The system is implemented in ROS using nodes. Two nodes were created to estimate the distance. One node is used to estimate the distance in the Carla simulator and is obtained experimentally, while the second node is used to estimate the distance in the real world and is based on distance estimation using the camera parameters and the reference width of the vehicle in the real world. The algorithm testing on the desktop computer was done using images from the simulator and the real world. It is noted that the algorithm runs at an average of 40 frames per second on the NVIDIA GTX1060 graphics card, which means that the algorithm works in real time. The developed algorithm allows detection of objects, even when only part of the object is inside the image or when more than one object overlaps. Vehicle detection from all positions around the vehicle is possible. It was noticed that the algorithm does not detect objects at a distance more than 50 meters, and that it detects objects at night and rain less often. It has also been noted that by increasing the distance, the precision of the estimated distance is reduced and certain adjustments have to be made to make the system better detection and precision for the distance estimation. The developed algorithm is easily customizable and easy to upgrade to detect other objects in traffic.

Keywords: object detection, ROS, Yolo, Yolo v2, distance estimation, Carla simulator, collision, autonomous vehicle

ŽIVOTOPIS

Mario Gluhaković rođen je 29.09.1995. godine u Tuzli, Bosna i Hercegovina. Godine 2010. upisuje JU Tehničku školu u Brčkom, Bosna i Hercegovina. U srednjoj školi se iskazuje na mnogobrojnim natjecanjima iz elektronike i elektrotehnike. U Osijeku 2014. godine upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. Godine 2017. stječe akademski naziv sveučilišni prvostupnik inženjer računarstva. Nakon toga upisuje diplomski sveučilišni studij automobilskog računarstva i komunikacija na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

Potpis:

PRILOZI

P3.1. – Označena baza podataka za treniranje Yolo v2 algoritma

P3.2. – OpenLabeling alat za označavanje slika

P3.3. – Python skripta za kreiranje train.txt i test.txt fajlova

P3.4. – Konfiguracijske datoteke za Yolo v2 algoritam

P3.5. – Implementacija algoritma u ROSu

P4.1. – Baza slika za testiranje Yolo v2 algoritma

P4.2. – Baza slika za testiranje čvora za procjenu udaljenosti u stvarnom svijetu

P4.3. – Baza slika za testiranje čvora za procjenu udaljenosti u Carla simulatoru